BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problems Mailbox.

Mar 30

WO 98/02831

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 17/30 — A1

(11) International Publication Number: **WO 98/02831**

(43) International Publication Date: 22 January 1998 (22.01.98)

(21) International Application Number: PCT/US97/11885

(22) International Filing Date: 10 July 1997 (10.07.97)

(30) Priority Data:
08/678,680 11 July 1996 (11.07.96) US

(71) Applicant: TANDEM COMPUTERS INCORPORATED [US/US]; 10435 N. Tantau Avenue, Loc 200-16, Cupertino, CA 95014 (US).

(72) Inventors: ERLENKOETTER, Ansgar; Auf der Heide 44, D-61267 Neu-Anspach (DE). DE BORST, Jeroen, Peter; Alte Mauergasse 5, D-61348 Bad Homberg (US). BONHAM, Peter, Douglas; Am Alten Bach 19, D-61352 Bad Homberg (DE).

(74) Agents: GRANATELLI, Lawrence, W. et al.; Graham & James L.L.P., 600 Hansen Way, Palo Alto, CA 94304 (US).

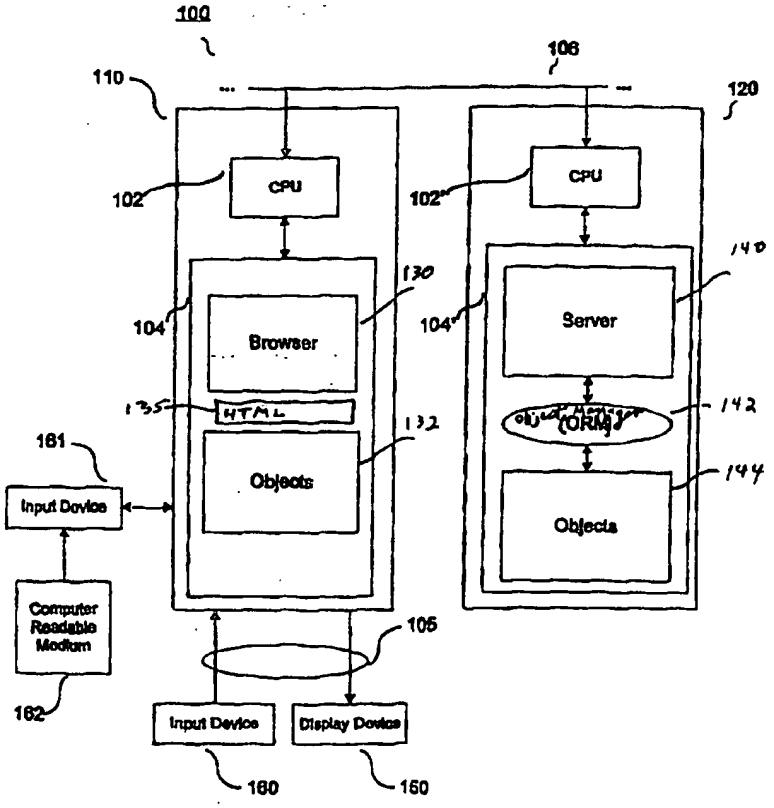
(81) Designated States: JP, European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: HYPERMEDIA OBJECT MANAGEMENT

(57) Abstract

A method and apparatus that uses a hypermedia approach to managing distributed objects. A first embodiment of the present invention uses the World Wide Web hypermedia system. A user initializes browser software that allows the user to browse and change various attributes of objects in the system. The browser communicates with a server that includes an http adapter and a gateway. The gateway can access objects in the system and generate HTML code in accordance with the objects. One embodiment of the present invention uses hierarchical tree-oriented objects. These objects are "self-describing" (also called "introspective"). The server queries the objects in response to the queries from the browser and each queried object responds with information about itself. In another preferred embodiment, the server initiates queries of the objects and retains this information for use in responding to later queries from the browser.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece			TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MR	Mauritania	UG	Uganda
BY	Belarus	IS	Iceland	MW	Malawi	US	United States of America
CA	Canada	IT	Italy	MX	Mexico	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NL	Netherlands	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NO	Norway	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	NZ	New Zealand		
CM	Cameroon			PL	Poland		
CN	China	KR	Republic of Korea	PT	Portugal		
CU	Cuba	KZ	Kazakstan	RO	Romania		
CZ	Czech Republic	LC	Saint Lucia	RU	Russian Federation		
DE	Germany	LI	Liechtenstein	SD	Sudan		
DK	Denmark	LK	Sri Lanka	SE	Sweden		
EE	Estonia	LR	Liberia	SG	Singapore		

HYPERMEDIA OBJECT MANAGEMENT

FIELD OF THE INVENTION

This application relates to object oriented programming and, in particular, to management of distributed objects via the World Wide Web.

BACKGROUND OF THE INVENTION

The past several years have seen an explosive growth of the use of distributed objects. Now, a single system may be composed of objects obtained from different vendors and having different interfaces. Such objects are called "heterogeneous objects." Thus, a system can be formed of a large and rapidly changing number of heterogeneous objects. Such a system requires a flexible and adaptive approach for system and application management. Conventionally, a heterogeneous system is managed by way of object-specific presentation facilities, i.e., by way of a user front-end that was written for each type of heterogeneous object. Such an approach is, however, too expensive in both development time and maintenance and administrative costs. In addition, conventional object management is often achieved through a single management center. Use of a single center is not efficient when a large number of objects need to be managed.

SUMMARY OF THE INVENTION

The present invention overcomes the problems and disadvantages of the prior art by using a hypermedia approach to object management. In this approach, each object is akin to a hypermedia document. The described embodiment of the present invention uses the World Wide Web hypermedia system. In a preferred embodiment of the present invention, a user initializes browser software that allows the user to browse and change various attributes of objects in the system. The browser communicates with a server that includes an http adapter and a gateway. The gateway can access objects in the system and generate HTML code in accordance with the objects.

A described embodiment of the present invention uses hierarchical tree-oriented objects. In a first embodiment, these objects are "self-describing"

(also called "introspective"). The server queries the objects in response to the queries from the browser and each queried object responds with information about itself. In another preferred embodiment, the server initiates queries of the objects and retains this information for use in responding to later queries from the browser.

In accordance with the purpose of the invention, as embodied and broadly described herein the invention is a system for managing objects, including a first server, comprising: a first receiver portion configured to receive a request in a hypermedia format; a first translator portion configured to convert the hypermedia request to an object request;a sender portion configured to send the object request to an object manager; a second receiver portion configured to receive a response from the object manager; and a second translator portion configured to convert the object manager response to the hypermedia format.

In further accordance with the purpose of this invention, as embodied and broadly described herein the invention is a method for browsing objects, where a browser communicates with a server, comprising the steps, performed by the browser, of: sending an initial URL to the server; receiving first data from the server, where the first data specifies an object corresponding to the URL; sending user-entered data associated with the object to the server; and receiving second data from the server, where the second data specifies a second object corresponding to the user-entered data.

Advantages of the invention will be set forth in part in the description which follows and in part will be obvious from the description or may be learned by practice of the invention. The advantages of the invention will be realized and attained by means of the elements and combinations particularly pointed out in the appended claims and equivalents.

BRIEF DESCRIPTION OF THE DRAWINGS

The accompanying drawings, which are incorporated in and constitute a part of this specification, illustrate several embodiments of the invention and, together with the description, serve to explain the principles of the invention.

Fig. 1 is a block diagram of a computer system in accordance with a preferred embodiment of the present invention.

Fig. 2 is another block diagram of a computer system in accordance with a preferred embodiment of the present invention.

Fig. 3 is a diagram of data sent between a browser, server, and object manager in accordance with the embodiment of Fig. 1.

Fig. 4 is a diagram of a format in which objects are organized.

Fig. 5 shows another example of a page displayed by the browser.

Figs. 6(a) and 6(b) show an example of HTML that causes the browser to display a portion of the page of Fig. 5.

Figs. 7(a) through 7(c) show further examples of HTML that result in the portions of page of Fig. 5.

Figs. 8(a) and 8(b) show several examples of ORM (Object Resource Management) requests made by the server to the object manager and the resulting responses from the object manager.

Fig. 9 shows another page displayed by the browser.

Fig. 10 shows layers of functions available to the object manager.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

Reference will now be made in detail to a preferred embodiment of the invention, an example of which is illustrated in the accompanying drawings. Wherever possible, the same reference numbers will be used throughout the drawings to refer to the same or like parts.

I. System Overview

Fig. 1 is a block diagram of a computer system 100 in accordance with a preferred embodiment of the present invention. Computer system 100 includes a first computer 110 and a second computer 120. First computer 110 and second computer 120 are connected together via line 106, which can be, for example, a LAN, a WAN, or an internet connection. Line 106 can also represent a wireless connection, such as a cellular network connection.

First computer 110 includes a CPU 102; a memory 104; input/output lines 105; an input device 160, such as a keyboard or mouse; and a display

device 150, such as a display terminal. First computer 110 also includes an input device 161 that reads computer instructions stored on computer readable medium 162. These instructions are the instructions of e.g., browser software 130. Memory 104 of first computer 110 includes browser software 130, Hypertext Markup Language (HTML) 135, and objects 132. A person of ordinary skill in the art will understand that memory 104 also contains additional information, such as application programs, operating systems, data, etc., which are not shown in the figure for the sake of clarity.

Second computer 120 includes a CPU 102' and a memory 104'. Memory 104' of second computer 120 includes server software 140, an object manager (ORM) 142, and objects 144. HTML 135 in the memory of first computer 110 was downloaded over line 106 from server 140 of second computer 120. A person of ordinary skill in the art will understand that memory 104' also contains additional information, such as application programs, operating systems, data, etc., which are not shown in the figure for the sake of clarity. Server 140, object manager 142, and objects 144 can also be located in memory 104 of first computer 110.

It will be understood by a person of ordinary skill in the art that computer system 100 can also include numerous elements not shown in the Figure for the sake of clarity, such as disk drives, keyboards, display devices, network connections, additional memory, additional CPUs, LANs, input/output lines, etc.

The following paragraphs provide a general discussion of the World Wide Web ("the Web"). The Web is built around a network of "server" computers, such as second computer 120, which exchange requests and data with each other using the hypertext transfer protocol ("http"). A human designer designs the layout of a Web page, which is then specified using HTML ("Hypertext Markup Language"). Several versions of HTML are currently in existence. Examples include HTML versions 2.0 and 3.0, as specified by the WWW Consortium of MIT. The HTML used in the described embodiment of the invention includes frames, forms, and tables, as are known to persons of

ordinary skill in the art.

A user views a Web page using one of a number of commercially available "browser" programs. The browser submits an appropriate http request to establish a communications link with a Web server of the network. A typical http request references a Web page by its unique Uniform Resource Locator ("URL"). A URL identifies the Web server hosting that Web page, so that an http request for access to the Web page can be routed to the appropriate Web server for handling. Web pages can also be linked graphically to each other.

Fig. 2 is an additional block diagram of a computer system in accordance with a preferred embodiment of the present invention. Browser 130 communicates with server 140. Server 140 includes an http adapter 202 and a management gateway 204. Http Adapter 202 handles communication via the known http protocol. Management gateway 204 communicates with object manager 142. Server 140 communicates with one or more objects 132, 144 using a request/response (RR) protocol, such as the ORM (Object Resource Management) protocol, which is discussed below. Note that objects 132 and 144 can be located on the same or different physical computers or machines. Server 140 also communicates with external interface 206, which communicates using the known SNMP and CMIP protocols. Server 140 also communicates with external gateway 208, which communicates using the known SNMP and CMIP protocols. The system can contain more than one servers 140 and more objects than are shown in Fig. 4.

Fig. 3 is a diagram of data sent between a browser, server, and object manager in accordance with the embodiment of Fig. 1. In the example of Fig. 3, the user has already begun execution of browser software 130. In step 302, the user enters the URL of server 140 by way of browser 130. The browser sends a request to the server and, in step 304, the server responds with the HTML to generate a home page. The home page allows the user to enter a URL (or to chose a URL from those known provided within the HTML of the home page). The user can then chose to set/browse objects in the system,

as described below. The user can also request information and statistics about once or more objects in the system.

In step 306, the user enters a URL of an object by way of browser 130. Server 140 converts the URL to a request to an object manager. For example, in the described embodiment, server 140 converts the URL to an ORM request, as described below. The ORM request is sent to the object manager, which returns object data in steps 308 and 310. Server 140 converts the object data into HTML, which is sent to browser 130 in step 312. The HTML may be based on a predetermined page template known to the server. Alternately, the format of a page may be determined "on the fly" based on the information obtained from the object manager. Server 140 converts all pathnames, such as object-links in the object data (see Fig. 4) to URLs in HTML and vice versa. Thus, if a user clicks on an area in a page displayed by the browser that corresponds to an object-link, browser 130 has the URL corresponding to that object-link. This new URL is sent to the server, which obtains the page information and sends HTML to display information for the object connected to the object-link.

Steps 314 through 320 represent a "set" mode, in which the user enters new values for an object by way of browser 130. In step 314, the user indicates that he wishes to enter "set" mode. This indication is usually accomplished by clicking on a button in the current page (thus, the HTML generated by server 140 should include HTML for this button). In step 316, server 140 sends a "form" for set mode. In step 318, the user enters new values into the form and clicks on "submit" (or "apply", (see Fig. 5), as is known to persons of ordinary skill in the art. Server 140 converts the submitted form to, for example, an ORM request, as described below. The ORM request is sent to the object manager, which returns object data in steps 317 and 319. Server 140 converts the object data of step 319 into HTML, which is sent to browser 130 in step 320.

Steps 322 through 332 represent a "browse" mode, in which the user views values associated with an object by way of browser 130. In step 322,

the user indicates that he wishes to enter "browse" mode. This indication is usually accomplished by clicking on a button in the current page (thus, the HTML generated by server 140 should include HTML for this button). In step 324, server 140 sends a "form" for browse mode. In step 326, the user enters new values into the form and clicks on "submit" (or "apply", see Fig. 5), as is known to persons of ordinary skill in the art. Server 140 converts the submitted form to, for example, an ORM request, as described below. The ORM request is sent to the object manager, which returns data corresponding to the object in steps 328 and 330. Server 140 converts the response of step 330 into HTML, which is sent to browser 130 in step 332.

II. Hypermedia Object Management

A. Object Organization

Fig. 4 is a diagram of a format in which objects are organized in a preferred embodiment. This organization is transparent to server 140 and browser 130. It will be understood that the present invention can be used with a number of object organizations and with a number of object management protocols. The embodiment described herein uses the ORM protocol, as described below.

The model of Fig. 4 assumes the following:

1) Management operations can be mapped to two basic operations: a) Get an attribute (or a set of attributes) of an object and b) set an attribute (or set of attributes) of an object.

2) All entities to be managed can be organized as a directed tree with nodes and leaves where the nodes are either (callable) objects or components (sub-parts of objects) with attributes as the leaves (with combined name/pair values), and

3) All knowledge about management operations and attributes is built into and controlled by the managed object.

Fig. 4 shows the following types of entities:

1) Objects

Objects encapsulate and control management aspects and respective

management operations. In the described embodiment, an object is identified by a "pathname," which is the destination for object calls. Each manageable object has its own virtual tree of components, attributes, and object-links.

2) Components

Components are the primary structuring mechanism within an object. Component sub-trees may be of arbitrary depth and component nodes may contain any number of object-links, other sub-components, or attributes.

3) Attributes

Attributes describe specific aspects of a component within an object (for example, "status=running" describes the state of a resource). Attribute nodes have additional properties beyond name and value, such s access mode and data type. Attribute nodes are leaves and do not have children.

4) Object-links

Object-links contain an object reference to a related object. As every object is responsible for its own virtual tree of resources, one object can provide a reference (hyperlink) to another object. Thus, in the described embodiment, a first object can have links to a second object, so that objects can be "walked" by way of browser 130.

5) Relations

Objects and components are the primary means for structuring and navigation in the described embodiment. Attributes have values that characterize the state of the resource. All operations (browsing and attribute retrieval/setting) are performed with respect to a single level of the tree (e.g., relative to a specific parent).

Server 140 preferably issues the following requests to object manager 142:

1) Get a list of linked objects,

2) Get a list of components and/or sub-components,

3) Get a list of attributes,

4) Set a list of attributes (Along with name/value pairs for each attribute), and

5) Get an extended list of attributes, which returns meta-information about the attribute, such as data type, allowed access mode (ro, rw) or valid ranges of new attribute values. Within the ORM model, all management operations are mapped to these five operations. Thus, every managed object preferably supports these five operations.

It should be understood that the attributes and object types shown in the examples herein are included only for the purposes of example. The present invention can be practiced using any appropriate object organization and type.

B. Server Interface

In the described embodiment, all messages passing in and out of server 140 are ASCII messages.

A example URL for object 402 of Fig. 4 would look like:

Http://ham/get/objectRoot/Component1/Component2/

A example URL for attribute 404 of Fig. 4 would look like:

Http://ham/get/objectRoot/Component1/Component2/Attr1/

In both of these URLs, "ham" stands for "HyperMedia Adapter to Management" and represents the address of server 140; "get" (this could also be "set") represents an operation to be performed on an object or attribute; and the remainder of the URL represents the tree of the object or attribute known to the object manager. Other URLs may also include additional information use, for example, by the object manager.

Fig. 5 shows a page displayed by browser 130 in "set" mode. Fig. 5 shows the values of attributes for a "Configuration" object component. These attributes include:

1) Status 520,

2) Maximum Concurrency 523,

3) Trace Level 524,

4) OSL Traces Enabled 526,

5) Script directory/Vol. 528,

6) Script File 530,

7) Cache Tcl Scripts 532,

8) Tcl Trace Enabled 534, and

9) Maximum Size of Synthesized Page 536.

Fig. 5 also shows an entry 522 for changing the status attribute. It should be understood that the attributes of Fig. 5 are presented for the sake of example only and are not to be taken in a limiting sense. Fig. 5 also shows a reset button 540 and an apply button 550. When the user clicks reset button, original attribute values are returned. When apply button 550 is clicked, browser 130 posts a form, as is known to persons of ordinary skill in the art.

Figs. 6(a) and 6(b) show an example of HTML generated by server 140. When browser 130 interprets the HTML of Fig. 6, it generates the portion containing attribute values 520-536 and buttons 540, 550 of Fig. 5. Figs. 7(a) through 7(c) show an example of HTML generated by server 140. When browser 130 interprets the HTML 702, 704, and 706 of Figs. 7(a) through 7(c), it generates portions 502, 506, and 504, respectively, of Fig. 5.

Fig. 9 shows another page displayed by browser 130 in accordance with HTML generated by server 140. The page of Fig. 9 is used to browse objects, but cannot change the attributes of objects.

The previous paragraphs discuss the browser GUI presented to the user and how server 140 translates between HTML and a protocol understood by the object manager. The following paragraphs describe the protocol used to communicate with object manager 142 about objects and to change objects in accordance with the HTML received by the server.

Figs. 8(a) and 8(b) show several examples of ORM requests made by the server 140 to object manager 142 and the resulting responses from object manager 142. Pages of the description shows formats of such requests and responses. Request 802 is an example of an OrmGet request sent from server 140 to object manager 142. The format of an OrmGet request is:

OrmGet: pathname

entity types

where pathname is a name of an object or an attribute. Possible entity types

are: "Object" (all known objects at this level), "Component" (a list of all components below the level of the path specified in the OrmGet), "Attribute" (a list of attributes for the current node; for every attribute, its name and "stringified value is returned; if the pathname already navigates to an attribute, the object manager returns the empty string), "Info" (returns "meta-attributes" such as mode, range and unit), and <none> (i.e., an empty string).

In request 802 of Fig. 8, the server "knows" about an object "HyperMedia Adapter NSK", possibly from receiving a URL from browser 130. Line 820 represents a version of the server (e.g., version 1.0). Line 822 is an "OrmGet" request for object "HyperMedia Adapter NSK". Server 140 requests information from object manager 142 about entity types (Info), Component, and Object (lines 824).

Response 804 is generated by object manager 142 and sent to server 140. The object has four components, no info, and no objects at the same level. As seen in step 312 of Fig. 3, server 140 generates HTML 604 of Fig. 7(c) in accordance with response 804 and sends the generated HTML to browser 130.

Assuming that the user wants to browse information about the Configuration component of object "HyperMedia Adapter NSK", browser 130 sends a request to server 140 to this effect. Server 140 then sends request 806 to the object manager, which responds with response 808. Request 806 is similar to request 802, but the pathname in line 830 is "HyperMedia Adapter NSK/Configuration".

Response 808 includes attributes for the "Info" entity. Thus, the response includes an attribute value, mode, field, and range for each of ten attributes of the component "Configuration". As seen in step 332 of Fig. 3, server 140 generates the HTML of Figs. 6(a) and 6(b) in accordance with response 808 and sends the generated HTML to browser 130 (see Fig. 5).

Assuming that the user wants to change one or more attributes of the Configuration component of object "HyperMedia Adapter NSK", browser 130 sends a request to server 140 to this effect (assuming that the browser is in

"set" mode). Server 140 then sends request 810 to the object manager, which responds by sending a status value (not shown).

A format of the OrmSet request is:

OrmSet: pathname

Attribute: name

Value: val

where "name" and "val" respectively, represent an attribute name and an attribute value. This command is shown in line 840. The command can include more than one Attribute/Value pairs.

In the example, request 810 specifies new values for eight attributes of component "Configuration." Assuming that no error occurs when the object manager changes the attribute values, server 140 generates HTML reflecting the new attribute values in accordance with the response and sends the generated HTML to browser 130 (not shown).

A preferred embodiment of the present invention has a server that interfaces with "self describing" (or "introspective") objects. The server sends requests to and receives responses from an ORM (Object Resource Manager). The system may include more than one ORM and more than one server. Each server may "know" about zero or more ORMs. Thus, the system is not centralized and does not necessarily depend on a central point to interface with the objects.

C. The Object Manager

1. Self Describing Objects

Fig. 4 shows an example of object organization in a preferred embodiment of the present invention. Pages of the description, shows examples of an ORM Server Support Library API (Application Program Interface) supported by the object manager to access objects in a preferred embodiment of the present invention. The routines in the API of pages are used by object manager (e.g., ORM 142 of Fig. 1) to receive requests from server 140 and to prepare responses to the requests. It will be understood be persons of ordinary skill in the art that any object manager can be used in

conjunction with the present invention, as long as the object manager is capable of communicating with server 140 and of fulfilling GetOrm and SetOrm requests from server 140.

Fig. 10 shows layers of functions available to the object manager. A Protocol layer 1002 handles the ORM protocol, e.g., decodes the request from server 140, initiates the corresponding functions, and assembles an ORM response. Protocol layer 1002 is the lowest layer and drives all calls to the upper layers by calling "registered" functions. A Node layer 1004 handles navigation between nodes, ie.e, parsing the pathname to locate the virtual node, which represents some management entity.

A Handle layer 1006 maps "virtual nodes" to real objects/data. Such a mapping results in a "handle." Handles are explicitly requested and released. An Aspects layer 1008 handles instances that are made up from more than one ORM tree. For example, the "statistics" Component is not a single Component n the tree, but is generated by the object manager. As another example, some attributes depend on others and cannot be modified independently, but have to be treated as a single, atomic operation. These groups of attributes within an instance are called "aspects" and the corresponding Aspect layer is provided to extract and modify groups of attributes within an instance.

An Attribute layer 1010 retrieves or updates a single attribute (of an aspect) and provides the meta information corresponding to this attribute. A Conversion layer handles the actual conversion of attributes between the external (ORM) and the internal (native) presentation. This layer also converts states and bitmaps to "friendly strings."

2. Web Agents

In another preferred embodiment of the present invention, the objects are not self-describing. In such an embodiment, one or more servers 140 in the system performs a "worm" function, i.e., one or more servers 140 follow object-links between objects and save all the information available concerning those objects. When a request is received from browser 130, server 140

sends its collected data to browser 130 (assuming that the collected data is not older than a threshold age value).

In summary, the present invention allows a user to manage objects by way of hypermedia, such as the world wide web. In a preferred embodiment, the objects are self-describing and respond to questions about themselves from one or more object managers. A server communicates with the object manager(s) and generates HTML from responses received from the object manager. Conventional browser software allows a user to indicate which objects he wishes to browse or change. Using a conventional hypermedia request/response protocol, the browser and server communicate to obtain information about objects and their attributes. The server also translates HTML/URLs received from the browser to requests to the object manager. Such a system allows a non-centralized object management system.

Other embodiments will be apparent to those skilled in the art from consideration of the specification and practice of the invention disclosed herein. It is intended that the specification and examples be considered as exemplary only, with a true scope of the invention being indicated by the following claims and equivalents.

5. Browse (readonly) and Change (read write) modes are differentiated by different URL's. For the Change mod an HTML input form is created with user interface elements dependant on the meta information provided with attributes. Dependant from this meta information, simple text input-or numeric input fields, popup boxes or radio buttons are generated.

6. A submit of this HTML form results in an HTTP POST request, which appears at the managements adapter with a special URL and the list of name/value pairs from the input form. These values can be checked against the information now retrieved from the object (see above) and a ORM SetAttribute request is send to the respective object. This object initiates the intended state changes or returns an error to the adapter, which then creates a new page, which reflects the outcome of the operation.

7. Access control can be either applied by the generic HTTP adapter, filtering POST methods for example or by the called object itself using principal identifiers in the ORM request.

5.3 Events and Alerts

Compared to SNMP or CMIP, ORM has no event or trap mechanism. With this respect it is much closer to the NSK management protocol called SPI. Instead event and alert support is provided by another mechanism (and object) within MSF, called *Alert Facility*, which is built on top of the *Common Execution Environment* (CEE).

6.0 ORM Protocol

6.1 General Characteristics

The ORM protocol is a simple request response protocol constructed out of lines of ASCII text and terminated by newline, to browse through the managed object and change its state. In this way, it is directly comparable to the HTTP protocol.

It is bytestream oriented, as it contains no length fields or has any fixed structure, but the individual items are separated by special characters, i.e. colons and newlines.

The logical end of a protocol unit is determined by an empty line i.e. a line with a newline character as the first character.

Remark: The decision for a pure ASCII protocol may be surprising, as MSF has a well defined presentation layer/protocol by IDL/PCU/GLU, but ORM is designed to support self contained object management, e.g. all manageable aspects of an object should be described by the managed object (or its manager) itself and just known by it. This implies, that a lot of human readable information has to be shipped with this protocol and using a pure ASCII protocol seemed quite natural. In addition the browsing nature of the protocol would have resulted in a quite unhandy IDL structure of unbounded sequences of any's or unbounded strings, probably with another layer of unbounded types above. Using ASCII strings simplifies the creation of protocol units. Last not least, ORM must also support the entities provided to build the MSF infrastructure itself, e.g. must also be available with and without these execution environments.

Internally the ORM protocol (ORM-P) is a tagged line protocol, as every line starts with a tag word followed by the tag value separated by a colon and terminated by an end of line character (newline). The protocol itself supports boxcarring (e.g. consolidated requests). Errors are reported inline with the data i.e., where they occur, by special error tags.

Request and response units are constructed the same way and the response is merely a "filled out" or "completed" request and as such, the information in it is self describing (i.e. need not necessarily correlated with the request). This is not true for authentication information, which is not mirrored in the response.

The ORM-P basically supports two kinds of operations:

1. get entities (OrmGet)
2. set attributes (OrmSet).

The two keywords OrmGet and OrmSet at the beginning of every sub-request describe the intended operation (i.e. are the operation tags).

In addition to these two basic operations there is a preamble required for every (consolidated) ORM packet, identifying the originators protocol version (tag OrmVersion) and in the case of a request, there is an optional tag, to pass authentication information from the client to the server (OrmWho).

6.2 Pathnames:

Pathnames provide the necessary navigation and identification information to locate a specific aspect of an object (or its internal entities). The different parts of the pathname usually are also the names of the entities in the generated user interface and as such, should be "friendly", descriptive names.

A pathname consists of a "/" separated list of components, which in turn may contain any printable character including whitespace.

For the current version, characters in the pathname components are restricted to the printable set of ASCII characters but may be extended to cover all printable ISO-8859-1 characters in the future (ORM-P is "8-bit clean", but this restriction is just to be more flexible in the choice of user interface construction).

There are two special component names, which follow the POSIX convention for filesystem tree navigation: "." and "..". These are only allowed in conjunction with an OrmPath tag and refer to the current node and to the parent of the current node respectively, where the "current node" must have been defined by a previous ORM sub-request in the same protocol unit (i.e. OrmGet or OrmSet). An attempt to traverse the parent of the root of the tree is treated as an erroneous pathname (compare with "cd" in a POSIX system).

6.3 Error Reporting (OrmError):

Errors are reported, where they are detected, i.e. the error tag (OrmError) usually appears directly behind the error causing tag in the response. This allows some kind of identification of the failed subrequest or protocol item even for large consolidated requests.

An error tag in the response also indicates, where the request was aborted. Previous subrequests in the same ORM packet may have been processed[1]. An error tag (OrmError) has a constant and a variable part, where the constant part identifies the kind of error, the variable part is used for additional hints, what caused the error. As the error is reported with a context, this context provides valuable information for error explanations.

The Format of an error protocol item:

```
OrmError:<decimal errorcode> <stringcode>:<variable part"\n"
```

6.4 Version Support: OrmVersion:

The first line in every protocol unit must identify the highest protocol version number, this protocol unit complies to. The actual version is 1.0!

1. When the ORM requests are issued via the IDL interface, the error also appears as an exception with the exception detail showing the actual ORM error code.

e.g.

```
OrmVersion:<major>.<minor>
```

A server/application may respond to this request with an error VersionMissmatch followed by its desired version id.

6.4.1 Protocol Conformance & Protocol Errors:

Unknown tags should be ignored unless they appear at a position, where another tag is required (an attribute tag must be followed by a value tag for example). If the latter occurs, the unidentified or unexpected tag is placed in the response message followed by an error tag with "ProtocolFailure"

6.5 Principal Identification Information: OrmWho

For the purpose of propagating the identification of a principal causing the request or to be used with the request, ORM-P supports a protocol item to ship any kind of (encoded) principal identifier with a request in the following way:

```
OrmWho:<principal identifier>
```

Up to now, neither the <id-scheme> nor the encoding or interpretation of the "encoded-principal-identifier" are specified in any detail by ORM-P but are up to the application and require an agreement between the ORM-P client and server. This may be subject of change!

6.6 Browse Operation: OrmGet

The operation tag "OrmGet" has to be followed by a colon (":" - the tag separator) and usually is followed by a "pathname". As every ORM protocol item, the operation tag followed by the optional pathname has to be terminated by a newline character. This first line is followed by a list of entity type specifyers requested.

6.6.1 Entity Types

The *OrmGet* request is followed by a list of type specifyers to describe the kinds of entities requested for browsing. Allowed types (and entity specifyers) are:

- *Object*

  requests a list of all known object links at this level. A friendly name and the link-address (NOR) is returned per object.

- *Component*

  A list of all components below the level of the path specified in the GET line is requested. A list of names is returned.

- *Attribute*

A list of attributes f r the current node is requested. For every attribute its name (*Name:<name>*) and its stringified value (*Value:*<valu) is returned. If the path in the OrmGet line already navigates to a single attribute, the returned name is the empty string.

- *Info* (implies Attributes)

  This type addresses the same type of entities as *"Attributes"*, but here meta information in addition to the name/value pair is requested. as there is Field for identifying the type of input expected, *Mode* to describe the access mode (read, read-write or write) and the so called hints (*Range* and *Unit*), which can be used by the user-interface to generate a more sophisticated presentation of the attribute. These *info* fields are described below in detail in the response section.

- <Empty>

  An empty type specifyer indicates that the validity of the pathname should be checked, but no information is requested.

As all ORM protocol items, every single type specifyer is terminated by a newline.

Examples:

```
1)OrmGet:/telnet/windows\n
  Component\n
  \n

2)OrmGet:/telnet/windows/#pty1/status\n
  Attribute\n
  OrmPath:../../pty2/status\n
  Attribute\n
  \n
3)OrmGet:/telnet\n
  Object\n
  Component\n
  Info\n
  \n
```

Note: Leading blanks in front of the tag or the tag value are ignored as well as lines, whose first character is a "#".

The OrmGet request may be called without a pathname specification (e.g. "OrmGet:\n") in which case the root object itself is referenced and the only valid type specifyer is "Object", which will return the "friendly" name of this object manger and its link address (Note: this link address may be another one, as the request was sent to, i.e. this allows redirecting the management requests to another object reference).

6.6.2 OrmGet Response

The response to a *OrmGet* request merely mirrors the request, but here the type specifyers are used as tags (to type the following entity) followed by the name of the entity (separated by ":") followed by a newline character. The name item is followed by a type dependent list of additional tagged items, describing further properties of this entity.

Although the partial responses below are listed per entity type, they appear in the same response unit in the same sequence as in the corresponding request unit, i.e. if the latter

requested entity types by the sequence "Object\nC mponent\Attribute\n", then the response will first list all available objects at this level followed by all available components followed by all available attribute/value pairs. If a requested type is not available at the specified level, an empty tag of that type (with a colon) is returned.

The response unit for an *OrmGet* request starts with the common response header (i.e. *OrmVersion:<version>*) followed by the "OrmGet: <pathname>" line followed by any number of the following constructs.

6.6.3 Object Entities:

```
Syntax:
<object-item> ::= "Object" [":" <string> "\n"
                  <object-link>] "\n"
<object-link> ::= "Link" ":" <string>

Example:

Object:Network Service Layer\n
Link:<obj-reference>\n
Object:Media Access Layer\n
Link:<obj-reference>\n
\n (if this is the end of the response!)
```

6.6.4 Component Entities: (have no additional properties)

```
Syntax:
<component-item> ::= "Component" [":" <string>] "\n"
```

Example:

```
Component:Configuration\n
Component:Statistics\n
\n (if this is the end of the response!)
```

6.6.5 Attribute Entities (simple request)

```
Syntax:

<attribute-item> ::= "Attribute" [":" <string> "\n"
                     <attribute-value>] "\n"
<attribute-value> ::= "Value" ":" <string>

Example:
      Attribute:Packets sent\n
      Value:1234\n
      Attribute:Packets received\n
      Value:4321\n
```

6.6.6 Attribute Entities (Info request):

```
Syntax:

<attribute-item> ::= "Attribute" [":" <string> "\n"
                        <attribute-info> | "\n"]
<attribute-info> ::= <attribute-value> "\n" (cnt)
                     <attribute-mode>"\n" (cnt)
                     <attribute-field> "\n" (cnt)
                    [<attribute-range> "\n"]
                    [<attribute-unit> "\n"]
<attribute-mode> ::= "Mode:" ["RO"|"WO"|"RW"]["P"]
<attribute-field>::= "Field:" <ORM-fieldtype>
<attribute-range>::= "Range:" <ORM-range-definition>
<attribute-unit> ::= "Unit:" <string>

Example:
Attribute:Status\n
Value:Running\n
Mode:RO\n
Field:String\n
Attribute:New Status\n
Value:Running\n
Mode:WO\n
Type:Enum\n
Range:Stopped,Aborted\n
....
```

Here the sequence of the different *info* tags is not relevant, but "Attribute" always starts a new property set for the next attribute. The "Range" and "Unit" tags are optional.

For "ORM-field type" and "ORM-range-definition" see below.

Errors & Exceptions specific to this request:

- NoSuchNode:

  The path specified does not point to a legal virtual node.This error is only reported immediately after a "Orm[Get/Set/Path]:<pathname>" command.

- InvalidOperation:

  The path specifies a node, which can not support the requested entity type, an "Attribute" request for an "Object" node or vice versa for example. This error is reported after a type specifyer, listing the type specifyer (without a colon) followed by a newline followed by the error tag.

6.7 Modification Requests: OrmSet

The set request starts with the set-tag "OrmSet:" followed by a pathname followed by a newline. The pathname at minimum must contain the name of the root e.g. "/<root-name>" (i.e. an empty pathname is not allowed!).

This line is followed by a sequence of line pairs containing the name of the attribute and its value, e.g.

```
Syntax:

<attribute-item> ::= "Attribute" [":" <string> "\n"
                          <attribute-value>] "\n"
<attribute-value> ::= "Value:" <string>

Example:
Attribute:New Status\n
Value:Suspended\n
Attribute:Reset Statistics\n
Value:Yes\n
\n (if this is the end of the protocol unit)
```

6.7.1 OrmSet Responses:

If no error occurred, the response to the *OrmSet* request is a copy of the request itself. Otherwise an error-tag may appear somewhere in the response, and if the underlying request/response protocol permits, the response is flagged with an error indicator.

The effect of an erroneous OrmSet sub-request is application dependant, but it is recommended, that a OrmSet sub-request either succeeds completely or has no effects at all (atomicity).

Any OrmSet requests preceding a failed one are not affected, any subsequent requests are ignored.

6.7.2 Error-Returns:

- NoSuchNode:

  The path specified does not point to a legal virtual node. This error is only reported immediately after a "SET:<pathname>" command.

- NoSuchAttribute:

  The string following an "Attribute:" tag does not identify a legal attribute. The error-tag follows the "Attribute:" tag (including the string).

- ValueOutOfRange

  The value specified is not within the range of legal new values for this attribute. The error tag follows the value-tag line.

- ValueInconsistent

  The set of attribute values in the request were no consistent or contradictory

- InvalidOperation:

  The designated Attribute is not writeable.

- NoPermission:

  The access rights of the requester do not allow to set the designated attribute.(This error and the previous one may have some overlap)

6.8 Request Type Independent Errors:

- ProtocolError:

  If pairs of tagged lines are expected and the sequence of pairs is not completed or an unknown or context illegal tag is detected, this error is generated, following the erroneous tagged line.

- InternalError:

  An internal error in the application/server was the cause, that this request could not be completed. (allocation failure, mangled structures). This indicates a severe error at the server side.

- BufferTooSmall:

  The response buffer specified is too small to return the full response.

- NoSpace:

  Some internal buffer could not be allocated or was to small for the requested operation.

6.9 ORM Attribute Info Descriptions:

Within the ORM protocol there are two ways to retrieve an attribute: the short form returns just the name of the attribute and its value and the long form, returning additional meta information for every attribute, which can be used to create reasonable user interface elements by the ORM client.

The following fields appear n the extended description:

- Field: identifies the kind of field, this attribute should be presented in
- Mode: identifies applicable operations (readonly, read/write, writeonly)
- Range: Provides hints for input checking and for user interface generation
- Unit: a free form field often describing the metric of the value or scale.

6.9.1 ORM Field Types:

Although in principal, the ORM field type item (*Field*) allows any principal character string, the ORM support library and the user interface generator (HTML synthesizer) will only support a limited set of predefined field types, to ease the presentation of attributes. If a field type is not recognized, the default "String" is assumed.

6.9.1.1 Field: Integer

The ORM protocol does not distinguish between unsigned and signed integers, e.g. every ascii string representing an integer may be prefixed by a "-" or a "+". There is also no size information in the field type. Any range restrictions have to be specified in the *Range* section.

Syntax:

Field:Integer

6.9.1.2 Field: Real

The field type *Real* identifies decimal floating point values. The allowed input formats are those of the POSIX 1003.2 scanf function for float and double values.

Syntax:

Field:Real

6.9.1.3 Field:HexOctet

This field type is used to display and enter binary data as pairs of hexadecimal character

Syntax:

Field:HexOctet (a sequence of hexadecimal digits)

6.9.1.4 Compound-Field Types:

The last set of field types allow much finer control of the input, an end user may provide to the ORM-P client side (or the client of the client...). These types are named *Enum* and *Set*, where *Enum* specifies a *"one out of m"* field and *Set* specifies a *"n out of m"* field.

Both types are only valid with an appropriate *Range* field in the hints section, where the possible alternatives must appear in a comma separated list.

These two types often transformed into "Pop-Up" menus (*Enum*) or option lists (*Set*) or similar by the user interface generator.

```
Syntax:

Field:Enum (single selection from "Range:")
Field:Set  (multiple choices from "Range:")
```

6.9.2 ORM Attribute Modes:

To generate reasonable user interfaces (as far as possible without object/component specific knowledge), the generator must know, whether an attribute is "read-only", "read-write" or "write-only". The latter is used to signal to the user interface, that this attribute should be only shown in "change-attribute" frames, if those are distinguished from pure browsing frames. An extension to these basic modes is provided for writable attributes to indicate, that an attribute value is mandatory, by appending the letter "M"

The different modes are simply encoded as two-letter strings followed by an optional "P", e.g.

```
Syntax

- RO        Read-Only
- WO[M]     Write-Only[non-null value mandatory]
- RW[M]     Read-Write[non-null value mandatory]
```

6.9.3 Range Identifiers:

The range identifier, tagged with "Range:" is used as a kind of hint (and therefore it is optional except for the compound fields) to the user interface generator, what kind of input/output field it should generate. In addition the information can be used to check any optional input and give the end-user appropriate responses or hints, if these input checks fail.

The range hints are type specific and as such different conventions are defined to specify valid ranges for an input field. The type independent convention is to separate alternatives by a comma "," and sequences by three subsequent dots "...".:

6.9.3.1 Range Specifications for Integer Fields:

Valid range specifications for the integer types are:

```
Range:1,4,8,16     valid: 1 of the values listed
Range:20...60      valid: all numbers between 20 and 60 including
Range:0...         valid: every integer including 0 (up to typemax)
Range:-20...+20    valid: every integer between -20 and +20 incl.
Range:             valid: every pos/neg integer within type
```

6.9.3.2 Range Specifications fro Floating Point Fields:

Valid range specifications for the real types are:

```
Range:0.1,0.5,0.8        valid: one of the values listed
Range:0.1e-3...0.1       valid: reals between 0.0001 and 0.1
```

6.9.3.3 Range Specifications for String Fields

If the first range value starts with a digit, the range indicates either the maximum or the range of valid string lengths. If the first character is non numeric, the range is interpreted similar to the compound *Enum* field below, i.e. one of these strings may be selected, but a different user interface element may be used (a list box). If the first character of the range string is a comma ",", this provided strings in the comma separated list are treated as examples, where the possible input is not restricted to the given alternatives. A major use of this kind of string selection is in file selection boxes.

```
Syntax:

Range:1...20              valid: strings with minimum length of 1
                          and max length of 20 characters
Range:10                  valid: a string with at max 20 characters.

Range:,file1.c,file2.c    file1.c or file2.c are valid options,but
                          other input is also valid
```

6.9.3.4 Range Specification for Compound Fields:

For the *Enum* and the *Set* type fields lists of alternatives are required in the range section. The comma separated list identifies the different options a user is allowed to select.

```
Syntax:

Range:<comma separated list of alternatives>

Example:

For Enum (choose one of)

Range:STOP,SUSPEND,ABORT

For field type Set (choose n or none of)

      Range:Trace IP,Trace UDP,Trace TCP
```

6.9.4 The Unit Specification:

The "Unit" specification is a free form string and currently not interpreted by the user interface generator. If present, it will append this string behind the value field as one would do with a unit description like "1.4 inches". Another important purpose of this field is for the use with customized object specific management pages (if used within an HTML environment). Here the unit could be used to identify an application specific type for example.

6.10 Navigation Request: Path

This request extends the previous operation (*OrmGet* or *OrmSet*) to a new subtree and follows these tags in its syntax. It may appear everywhere, where a *OrmGet* or *OrmSet* tag may appear, except that it must be preceded by one of these items in the same protocol unit. It usually is only found in sequences of ORM statements resulting from a "Dump" request!

Syntax:

```
OrmPath:<pathname>
```

Semantics: Extends the previous *OrmGet* or a *OrmSet* request into another subtree within the same object.

6.11 Summary of ORM Error Codes:

NoPermission: 1

The current authentication can not be used to perform the requested operation

NoSuchNode: 2

The pathname specified in a Orm[Get/Set/Path] request does not point to a known

node.

NoSuchAttribute: 3

The attribute specified in a OrmSet request could not be found

NoSuchObject: 4

The Object specified could not be found.

InvalidOperation: 5

The operation requested is not valid for this type of entity. (Example: attribute is not writable, request components of an attribute)

ProtocolFailure: 6

A sequence of tags was encountered, which could not be parsed and decoded.

VersionMissmatch: 7

The object could not deal with the version of the request packet.

CommunicationError: 8

This is a client side error to map lower level communication errors too, if necessary.

ValueOutOfRange: 9

The value passed in with a set request for an attribute is not within the allowed range and could not be accepted.

ValueInconsistent: 10

The combination of values passed with a set request is not acceptable.

NoSpace: 12

The request could not be completed because of internal space restrictions in the object.

BufferTooSmall: 13

The response to the request exceeds the size of the response buffer provided by the underlying protocol.

InternalError: 14

ApplicationError: 15

These two errors are used to report back implementation problems like corrupt data structures, where the InternalError usually is generated by the ORM support library, the ApplicationError instead is issued by the higher "application" layers.

3.1 ORM Protocol Layer And Upcall Interfaces

This section was generated from *<stdin>* by CDOC on Sun Jan 29 17:00:50 1995.

ORM Application Context

Application Server Capsules may serve different kind of requests and therefor may have multiple domains of objects to be managed listening on multiple ports. Following the ORM model, this may result in multiple parallel independant trees.

The ORM parser supports this by maintaining an application context, which has to be passed to the protocol layer to handle a request (there is also an opaque *call-context*, which may be passed to the protocol layer, but this isn't interpreted by the ssl).

The application context contains beside (an opaque pointer) to the (virtual) root of the virtual tree, mainly a list of tree/application specific function pointers. Before the first request can be passed on to the ORM protocol layer, this context has to be established with the ORM SSL via a call to ORM_ContextInitialize.

Accordingly there exists a function to inform ORM that this application context is not needed anymore (release).

The following lists the function prototype definitions for actual functions to be provided, when establishing a context.

Note: Some functions are defined to return pointers to character strings (ORM_String). If the ORM protocol handler is used it is guaranteed, that the same function will not be called, before the string is copied or otherwise not needed anymore. This allows the use of a single private string buffers per function, if necessary.

3.1.1 Authentication

The following list of functions are included to enable an application to maintain its own authenticated context. The ORM protocol just allows to forward some authentication related information from the client to the server (WHO...). This is passed on to the application layer as is, if encountered by the parser. The actual meaning of this data is application and user interface dependant.

3.1.2 Function Type ORM_AuthenticateFunc

Performs any necessary authentication or preparation of authentication structures. Usually, the authentication information is used to setup some context in the call-context, which is passed to the node/handle layer upcalls. It is up to the application layer to free/clear such context after return from the protocol layer.

Declaration:

```
typedef IPM_Status (*ORM_AuthenticateFunc) (
                ORM_AppCallContextDef   callcontext,/* in*/
                ORM_String              authstring  /* in */
```

Fields:

callcontext	An opaque pointer to any kind of context, the caller has established.This passed to the node and handle layer.
authstring	The string, the client passed in his request, if any. Usually uid:passwd
status	ORM_ENoError: if successfull, ORM_EPermissionDenied, if authentication unknown.

3.1.3 ORM_AuthFuncDef

This structure is used to pass the Authentication function to ORM_ContextInitialize

Declaration:

```
typedef struct ORM_AuthFuncTag {
    ORM_AuthenticateFunc        auth;
} ORM_AuthFuncDef;
```

3.1.4 Virtual Node & Tree Function Types

The following list of functions (function types) are used to access the virtual tree of components, attributes and linked objects. They usually don't deal with application specific data.

3.1.5 Function Type ORM_NodeLookUpFunc

This is the central function for the traversal of *the* tree .

Returns an opaque pointer to a virtual node, which may subsequently be called to retrieve properties or children of specific types.

Declaration:

```
typedef ORM_Status (*ORM_NodeLookUpFunc ) (
        ORM_AppCallContextDef   callcontext,/* in */
        ORM_AppNodeDef          root,       /* in */
        ORM_String              pathname,   /* in */
        ORM_AppNodeDef          *node,      /* out */
        ORM_NodeTypeDef         *nd_type    /* out */
    );
```

Fields:

callcontext	is an opaque pointer to the application specific call context provided with the Do_Request function.
root	Opaque Pointer to root of virtual tree. This may be NULL, and is taken from the application context.
pathname	is a / separated list of component names optionally preceded by the name of the object (e.g. if the first component matches the roots object name, strip it, else take the first component to be a child under the applications root). Support for *un*x* style directory navigation . and .. is highly recommended/required. A pathname of .. applied to the root with request type *Object* should return the root name and the actual servers link address (NOR)
node	The opaque node pointer, if found
nd_type	The ORM_NodeType of the node found
return	ORM_ENoError in case of success, or any other ORM error in case of failure.

3.1.6 Function Type ORM_NodeChildNextFunc

Used to subsequently scan the children of a single parent. Returns the next child of type *type* of parent *parent*, which logically follows the child returned by the previous call to *NodeChildNext()*, now passed in as *lastchild*. E.g. If *lastchild* is set to *NULL* the logically first child of this parent is requested. If there are no children (of the requested type), then NULL must be returned with ORM_Status set to ORM_NoError.

Declaration:

```
typedef ORM_Status (*ORM_NodeChildNextFunc ) (
    ORM_AppCallContextDef   callcontext,/* in */
    ORM_AppNodeDef          parent,     /* in */
    ORM_AppNodeDef          lastchild,  /* in */
    ORM_NodeTypeDef         type,       /* in */
    ORM_AppNodeDef          *child,     /* out */
    ORM_String              *name       /* out */
    );
```

Fields:

callcontext	is an opaque pointer to the application specific call context provided with the Do_Request function.
parent	Opaque pointer to the virtual parent node.
lastchild	Opaque pointer to the last child returned by a call to this function (in this request), or NULL to request the first child.
type	The type of entity, which is requested (ORM_ObjectType, ORM_ComponentType, ORM_AttributeType or ORM_AnyType).

node	Pointer where to store the reference to the node found
name	Pointer to name of node found.
returns	status value. Possible status values, see below!

3.1.7 Function Type ORM_NodeChildByNameFunc

The little sister of ORM_NodeLookUp. Looks for a child with name *childname* directly under the given parent *parent*. This function is primarily used within the processing of Set- Attribute requests. If there is no child with this name, return NULL and an error status (see below)

Declaration:

```
typedef ORM_Status (*ORM_NodeChildByNameFunc ) (
        ORM_AppCallContextDef   callcontext,/* in */
        ORM_AppNodeDef          parent,     /* in */
        ORM_String              childname,  /* in */
        ORM_AppNodeDef          *child,     /* out */
        ORM_NodeTypeDef         *child_type /* out */
```

Fields.

parent	Opaque pointer to the virtual parent node.
childname	Name of the child (attribute), i.e. every printable char except '/'
child	Pointer where to store the reference to the node found
child_type	Pointer to type of node found.
returns	ORM_ENoError if child was found, else ORM_ENoSuchNode.

3.1.8 Function Type ORM_NodeTypeGetFunc

returns the type (enum ORM_NodeTypeDef) of the given node.

Declaration:

```
typedef ORM_NodeTypeDef (*ORM_NodeTypeGetFunc ) (
    ORM_AppNodeDef      node        /* in */
    );
```

Fields:

node	a pointer to a virtual node
returns	a valid type or ORM_NodeTypeUnknown.

3.1.9 Function Type ORM_NodeNameGetFunc

returns the name (ORM_String) of the given node.

Declaration:

```
typedef ORM_String (*ORM_NodeNameGetFunc ) (
    ORM_AppNodeDef      node        /* in */
    );
```

Fields:

node	a pointer to a virtual node
returns	a valid null terminated string of characters or NULL

3.1.10 Function Type ORM_NodeNotFoundTrapFunc

This function is kind of special by providing the application layer a chance, if the lookup of a node failed,to create that node.

Normally, referencing a non-existent node in the pathname of an ORM request is treated as an error, except this is an internal ORM restore request. Reloading an ORM tree into an application may encounter subtrees, which where dynamically created by the application during a previous run (usually via a *New* subtree).

This function is totally application dependant and is not covered by the ORM-SSL other than via this hook.

Declaration:

```
typedef ORM_Status (*ORM_NodeNotFoundTrapFunc) (
    ORM_AppNodeDef      parent,     /* in */
    ORM_String          name,       /* in */
    ORM_RequestTypeDef  request,    /* in */
    ORM_AppNodeDef      *newnode    /* out */
    );
```

Fields:

parent	Reference to parent node
name	Name of node not found under this parent.
request	Kind of ORM request (get/set/dump/restore) causing this lookup failure.
newnode	Where to store the reference to the new node, if one was created.
returns	ORM_ENoError, if a node with the given name was created else ORM_ENoSuchNode.

3.1.11 Structure ORM_NodeFuncDef

This structure bundles the virtual tree related functions for passing to ContextInitialize

Note: The ORM_NodeNotFoundTrapFunc is not included in this function array, because it is application special anyway and must be passed explicitly, see ContextInitialise()

Declaration:

```
typedef struct ORM_NodeFuncTag {
    ORM_NodeLookUpFunc          lookup;
    ORM_NodeChildNextFunc       childnext;
    ORM_NodeChildByNameFunc     childbyname;
    ORM_NodeTypeGetFunc         typeget;
    ORM_NodeNameGetFunc         nameget;
} ORM_NodeFuncDef;
```

3.1.12 Application Handles

The following two function types are used to link the virtual nodes in the tree to (parts of) actual application data instances, visible to the ORM support layer as opaque handles. When an application handle is requested from the application layer, *real things* happen to start and it is assumed, that the instances are valid and available, until explicitly released by the ORM layer. The handles together with the aspect (identifying the type of handle to the application) will be passed to the application specific functions, when actual values have to be accessed (either for *get* or *set*). If these functions are not set in the ORM context, NULL will be passed into those calls for both, the handle and the handleclass.

3.1.13 Function Type ORM_HandleGetFunc

Request (and lock) an actual handle (pointer to an application level instance) and a handleclass based on the current virtual node and the current principal.

Declaration:

```
typedef ORM_Status (*ORM_HandleGetFunc) (
    ORM_AppCallContextDef   callcontext,/* in */
    ORM_AppNodeDef          node,       /* in */
    ORM_RequestTypeDef      op,         /* in */
    ORM_AppHandleDef        *handle,    /* out */
    ORM_AppAspectDef        *aspect     /* out */
    );
```

Fields:

callcontext	is an opaque pointer to the application specific call context provided with the Do_Request function.
node	Pointer to current Node.
op	Operation Code, e.g. ORM_Request
handle	Pointer, where to store the handle reference

aspect	Pointer, where to store the aspect r ference
returns	ORM_ENoError if no error occured r any of the ORM error codes.

3.1.14 Function Type ORM_HandleReleaseFunc

Returns a given handle back to the application layer. This should be more understood as an *unlock* operation than a free!

Declaration:

```
typedef void (*ORM_HandleReleaseFunc ) (
    ORM_AppCallContextDef   callcontext,/* in */
    ORM_AppHandleDef        handle,     /* in */
    ORM_AppAspectDef        aspect,     /* in */
    ORM_RequestTypeDef      op          /* in */
    );
```

Fields.

callcontext	is an opaque pointer to the application specific call context provided with the Do_Request function.
handle	a handle obtained via a call to HandleGet
aspect	Aspect as returned from HandleGet
op	Operation Code, e.g. ORM_Request

3.1.15 Function Type ORM_ObjectLinkGetFunc

Retrieve the Object Link from a node of type Object given the node, the handle and the aspect. The standard Handle Layer functions just return the link stored in the corresponding field in the node struct.

Declaration:

```
typedef ORM_Status (*ORM_HandleObjectLinkGetFunc ) (
    ORM_AppNodeDef      node,       /* in */
    ORM_AppHandleDef    handle,     /* in */
    ORM_AppAspectDef    aspect,     /* in */
    ORM_ObjectLink      *link       /* out */
    );
```

Fields:

node	Reference to node of Object Type
handle	Reference to application defined handle as returned from HandleGet
aspect	Reference to application defined aspect as returned from Han-

link	Location where to store the reference to the stringified link information
returns	ORM_ENoError if successfull, else ORM_InvalidOperation, if the node is not of type Object

3.1.16 Function Type ORM_AttributeDescrGetFunc

Retrieve the opaque reference unique to a node of type Attribute (usually the attribute descriptor), given the node, the handle and the aspect. The standard Handle Layer functions just return the pointer stored in the corresponding field in the node struct.

Declaration:

```
typedef ORM_Status (*ORM_HandleAttributeDescrGetFunc ) (
    ORM_AppNodeDef          node,       /* in */
    ORM_AppHandleDef        handle,     /* in */
    ORM_AppAspectDef        aspect,     /* in */
    ORM_AppAttributesDescrDef *attribdesc /* out */
    );
```

Fields:

node	Reference to node of Object Type
handle	Reference to application defined handle as returned from HandleGet
aspect	Reference to application defined aspect as returned from HandleGet
attribdesc	Location where to store the reference to the attribute information
returns	ORM_ENoError if successfull, else ORM_InvalidOperation, if the node is not of type Object

3.1.17 Structure ORM_HandleFuncDef

This structure bundles the handle related functions for passing to ContextInitialize

Declaration:

```
typedef struct ORM_HandleFuncTag {
    ORM_HandleGetFunc                get;
    ORM_HandlePutFunc                release;
    ORM_HandleObjectLinkGetFunc      link;
    ORM_HandleAttributeDescrGetFunc  attrib;
    } ORM_HandleFuncDef;
```

3.1.18 Accessing Application Data: Aspects

the following group of functions (function types) has to be provided to access actual values of the application either for retrieval or for updating. All functions in this group are mandatory, if the ORM protocol layer is used.

3.1.19 Function Type ORM_AspectCallGetFunc

This function retrieves an *aspect* from the application layer, e.g. a reference to a blob of native application data (a pointer to a (part of) an application data structure, or a response buffer ....). The ORM protocol layer calls this function once for every unique handle/aspect combination (and not per Attribute) within a single AttributeGet Request. If the HandleGet Function returns a different pair or there are no more attribute nodes to process, the current aspect is released!

Declaration:

```
typedef ORM_Status (* ORM_AspectCallGetFunc) (
    ORM_AppHandleDef     handle,      /* in */
    ORM_AppAspectDef     aspect,      /* in */
    ORM_AppDataPtrDef    *current     /* out */
    );
```

Fields:

handle	Handle as retrieved from HandleGet
aspect	Aspect Reference, as retrieved from HandleGet
current	Where to store the reference to the current value (opaque)

3.1.20 Function Type ORM_AspectCallInitFunc

This function requests an *aspect* container from the application layer, e.g. a reference to a blob, where new attribute values can be selectivly written to to perform AttributeSet requests. In addition hte application layer may return a reference to the current aspects values (cmp CallGet), which is passed unchanged to the CallSet routine. The ORM protocol layer calls this function once for every unique handle/aspect combination (and not per Attribute) within a single AttributeSet Request. If the HandleGet Function returns a different pair for a node or there are no more attribute nodes to process, the CallSet function is called (Note: AspectRelease is only called for aspects retrieved via CallGet!) The

ORM SSL Implementation of these functions copies the current values and returns a reference to this copy in *new* and a reference to the current values in *current*.

Declaration:

```
typedef ORM_Status (* ORM_AspectCallInitFunc) (
    ORM_AppHandleDef    handle,     /* in */
    ORM_AppAspectDef    aspect,     /* in */
    ORM_AppDataPtrDef   *new,       /* out */
    ORM_AppDataPtrDef   *current    /* out */
    );
```

Fields:

handle	Handle as retrieved from HandleGet
aspect	Aspect Reference, as retrieved from HandleGet
new	Where to store the reference to the native blob to update with new attribute values (opaque)
current	Where to store the reference to the current aspect (opaque)

3.1.21 Function Type ORM_AspectCallSetFunc

This function is called to actually apply the new attribute values for the current aspect by the application layer. It is up to the aspect/application layer, to check the values in the request structure for validity and consistency and to determine which attributes got new values (by comparison with the *current* values). In addition it is the responsibility of the aspect/application layer to deallocate any structures allocated by AspectCallInit. Only if the Set-Function is not called, the call to AspectRelease is performed.

The ORM protocol layer calls Set-function once for every unique handle/aspect combination (and not per Attribute) within a single AttributeSet Request. If the HandleGet Function returns a different pair for a node or there are no more attribute nodes to process, the CallSet function is called (Note: AspectRelease is only called for aspects retrieved via CallGet!) The ORM SSL Implementation of these functions copies the current values and returns a reference to this copy in *new* and a reference to the current values in *current*.

Declaration:

```
typedef ORM_Status (* ORM_AspectCallSetFunc) (
    ORM_AppHandleDef    handle,     /* in */
    ORM_AppAspectDef    aspect,     /* in */
    ORM_AppDataPtrDef   new,        /* in */
    ORM_AppDataPtrDef   current,    /* in */
    ORM_String          *reDetail   /* out */
    );
```

aspect	Aspect Reference, as retrieved from HandleGet
request	Where to store the reference to the native blob to update with new attribute values (opaque)
current	Where to store the reference to the current aspect (opaque)
rsdetail	Where to store a textual hint, why the call failed, if any.
returns	ORM_ENoError if new values could be applied successfully, else ORM_ERange.

3.1.22 Function Type ORM_AspectReleaseFunc

Used to tell the application layer, that the reference retrieved via an AspectGet or AspectInit call is no longer needed anymore by the ORM layer. This function is called, when GetHandle returns a new handle aspect call within a AttributeGet processing or a conversion in an AttributeSet processing failed.

Declaration:

```
typedef void (* ORM_AspectReleaseFunc) (
    [illegible]     handle,     /* in */
    [illegible]     aspect,     /* in */
    [illegible]     current,    /* in */
    ORM_RequestTypeDef  reqtype     /* in */
    );
```

Fields:

handle	Handle as retrieved from HandleGet
aspect	Aspect Reference, as retrieved from HandleGet
current	Reference to data as returned from AspectCallInit or AspectCallGet.
reqtype	ORM_RequestGet or ORM_RequestSet depending whether this dataptr resulted from an AspectGet or AspectInit call.

3.1.23 ORM_AspectFuncDef

This function groups the function pointers of the aspect layer

Declaration:

```
typedef struct ORM_AspectFuncTag {
    ORM_AspectCallGetFunc   callget;
    ORM_AspectCallInitFunc  callinit;
    ORM_AspectCallSetFunc   callset;
    ORM_AspectReleaseFunc   release;
} ORM_AspectFuncDef;
```

3.1.24 Attribute Functions

The following group of functions is called to actually perform the the single attribute Get/Set and the corresponding conversions between the applications native and the ORM (ascii) presentation.

3.1.25 Data Structure: ORM_AttributeInfoDef

This structure is used to return the all the meta information and the actual value of an attribute. It is passed by reference to the application/attribute layer to be filled. Note: The string pointers do not point to valid buffers, when passed to the attribute layer!

```
<mode>  R|(R), W|(R) RW|R|
<type>  INT,[ 1..4 ], REAL[32|64], STRING, HEXOCT,
        [R|L]N, MCH|ICE
<value> the current value in its ascii presentation
<range> Optional: The range string
<unit>  Optional: The unit string
```

Declaration:

```
typedef struct ORM_AttributeInfoTag  {
    ORM_string          value;
    ORM_[illegible]     mode;
    [illegible]         field;
    ORM_[illegible]     range;
    ORM_string          unit;
} *ORM_AttributeInfoDef;
```

3.1.26 Function Type ORM_AttributeNativeToStringFunc

This function converts the applications native value of an *attribute*, specified by *handle*, *aspect* and the attribute descriptor to a C-string (ORM_String).

Declaration:

```
typedef CRM_Status (*CRM_AttributeNativeToStringFunc)(
        CRM_AppHandleDef        handle,         /* in */
        CRM_AppAspectDef        aspect,         /* in */
        CRM_AppAttribDescrDef   attribdescr,    /* in */
        CRM_AppDataPtrDef       dataptr,        /* in */
        CRM_String              *strvalue       /* out */
        );
```

Fields:

handle	Handle as obtained from the last call to HandleGet or NULL.
aspect	Aspect as returned from the last call to HandleGet or NULL
attribdescr	*Attribute Descriptor as returned form AttribDescrGet* call.
dataptr	Opaque Pointer as returned from AspectGetCall.
strvalue	Where to store the reference to the converted value.
returns	ORM_ENoError (Null) if conversion was successfull, else a valid ORM Error return code.

3.1.27 Function Type ORM_AttributeNativeToInfo

This function performs the same as the previous function *ORM_AttributeNativeToString, except that it also provides* the additional meta information to this attribute, as far as available.

Declaration:

```
typedef CRM_Status (*CRM_AttributeNativeToInfoFunc)(
        CRM_AppHandleDef        handle,     /* in */
        CRM_AppAspectDef        aspect,     /* in */
        CRM_AppAttribDescrDef   attribdescr,/* in */
        CRM_AppDataPtrDef       dataptr,    /* in */
        CRM_AttributeInfoDef    info        /* in, indirect out */
        );
```

Fields:

handle	Handle as obtained from the last call to HandleGet or NULL.
aspect	Aspect as returned from the last call to HandleGet or NULL
attribdescr	*Attribute Descriptor as returned form AttribDescrGet* call.
dataptr	Opaque Pointer as returned from AspectGetCall.
extref	Pointer to structure, where to store the string references.

returns ORM_ENoError (Null) if conversion was successfull, else a valid ORM Error return code.

3.1.28 Function Type ORM_AttributeStringToNativeFunc

This function converts an ORM_String value for an attribute into the applications native presentation. The conversion should be done into the structure (dataptr) obtained by a call to AspectCallInit().

Declaration:

```
typedef ORM_Status (*ORM_AttributeStringToNativeFunc)(
        ORM_AppHandleDef        handle,     /* in */
        ORM_AppAspectDef        aspect,     /* in */
        ORM_AppAttribDescrDef   attribdescr,/* in */
        ORM_AppDataPtrDef       dataptr,    /* in, indirect out */
        ORM_String              strvalue    /* in */
        );
```

Fields:

handle Handle as obtained from the last call to HandleGet or NULL

aspect Aspect as returned from the last call to HandleGet or NULL

attribdescr *Attribute Descriptor as returned form AttribDescrGet* call.

dataptr Opaque Pointer as returned from AspectGetCall.

strvalue New value as a C-String (ascii).

returns ORM_ENoError (Null) if conversion was successfull, else a valid ORM Error return code.

3.1.29 Structure ORM_AttributeFuncDef

This structure bundles the attribute related functions for passing to ContextInitialize

Declaration:

```
typedef struct ORM_AttributeFuncTag {
    ORM_AttributeStringToNativeFunc     stringtonative;
    ORM_AttributeNativeToStringFunc     nativetostring;
    ORM_AttributeNativeToInfoFunc   infotostring;
    } ORM_AttributeFuncDef;
```

3.1.30 Structure ORM_ContextDef

This is an internal structure to ORM and opaque to the application layer. It stores the function pointers and the information of the root node.

Note: This structure and the related procedure definitions may change

authfuncs	Pointer to list of authentication related functions or NULL, if no application specific authentication is needed.
notfound	No description

3.1.32 ORM_ContextRelease

Release an Application Context.

Prototype:

```
void
ORM_ContextRelease( ORM_ContextDef contxt);
```

Parameters:

contxt	Pointer to application context as obtained from ORM_ContextInitialize

3.1.33 ORM_DoRequest

This function calls the protocol layer to parse an ORM request received and act on it accordingly via upcalls to functions in the application context, i.e. this is the function to be dispatched, when ORM requests are received on a server port.

Prototype:

```
ORM_Status
ORM_DoRequest(
    ORM_ContextDef      appctxt,
                        *callctxt,
    ORM_requestDef      request,
    long                reqlen,
    ORM_ResponseDef     response,
    long                *maxresp
    );
```

Parameters:

appctxt	The application context reference as returned from ORM_ContextInitialize.
callctxt	An arbitrary call context (reference) maintained by the application layer and passed to the authentication, node and handle upcalls.
request	Pointer to received ORM request
reqlen	Length of request buffer in bytes
response	Pointer to allocated response buffer

maxresp	Reference to maximum response buffer length in bytes, on return, points to number of bytes used in response buffer

3.2 ORM Node Layer

This section was generated from *<stdin>* by CDOC on Fri Jan 27 19:59:34 1995.

The ORM Node layer adds another level of ORM application/server support, as it actually maintains a tree structure to access the application level datastructures.

This level is accessed from the application/server level via the *ORM_Node...* functions to actually build/destroy the tree of objects, components and attributes.

On the other side it is called from the protocol level and frees up the application to provide the appropriate functions for navigation and name space/entity management itself.

3.2.1 Application Handles

The nodes of the node layer provide a tree structured view to application/server level data, but they (usually) do not contain the actual data. A link to the actual instances of application level data is maintained by *handles* and *aspects*. Both are opaque to the ORM-Node level but are interpreted at the layer on top of ORM-Node. Typically the handle is a pointer to some application level instance, and the aspect is a pointer, index or type identifier, which identifies the type of the instance

3.2.2 The ORM_Node Structure

Instances of this structure maintain the tree of virtual components, objects and attributes

Every node has a name and a type, identifying the three different entity types: Object, Component or Attribute. Object and Attribute nodes are leaf nodes, e.g. they can't have children.

In addition, every node has a parent and a next pointer, to link the actual tree structure. Only component nodes have a pointer to the list of children.

Object Nodes have an additional attribute, called the Link (or Link-Info which usually is a stringified NOR).

Attribute Nodes reference a single attribute by, which is characterize by additional information like

- a value type, which describes the kind of value e.g. integer (different sizes), real (sizes!), string, *single-selection* or *multiple choice*
- a value mode, specifying this attribute as read-only read-write, write-only or persistant.
- *hints* section, which contains additional information for use by the user-interface creator, e.g. valid ranges for this value and a unit string. Both values are optional.

The nodes provide a tree structured view to application/ server level data, but they (usually) do not contain the actual data.

3.2.3 Struct NodeDef

Declaration:

```
typedef struct ORM_NodeTag {
    ORM_NodeTypeDef      type;
    short                flag;
    char                 *name;
    struct ORM_NodeTag   *parent;
    struct ORM_NodeTag   *next;
    ORM_AppHandleDef      handle;
    ORM_AppAspectDef      aspect;
    union {
        struct
            struct ORM_NodeTag *first;
            struct ORM_NodeTag *last;
            } comp;
        struct
            void     *desc;
            } attrib;
        struct {
            char     *link;
            } threat;
        } u;
} *ORM_NodeDef;
```

Fields:

type	indentifies the type of entity, this node describes, i.e. ORM_NodeType[Object, Component Attribute, Unknown]
flag	Internal use
name	The name of the node (object, component or attribute name
parent	pointer to the parent in the tree, NUL for the root of the tree.
next	pointer to next sibbling in chain. This defines the order in which nodes of a given type appear in the response
handle	an opaque pointer for use by the upper layers
aspect	another opaque identifier for use by the upper layers
u.comp	union variant for component nodes
u.comp.first	pointer to first child of this component node
u.comp.last	pointer to last child of this component node
u.attrib	union variant for attribute nodes
u.attrib.descr	opaque pointer for use by upper layers

u.object.link pointer to stringified link-address of this object (NOR), e.g. the *hyperlink*

3.2.4 ORM_NodeCreate

Creates a new unlinked node. Usually only used by convenience functions and to create the root node.

Prototype:

```
ORM_NodeDef
ORM_NodeCreate( ORM_String       name,    /* in */
                ORM_NodeTypeDef type     /* in */
              );
```

Parameters:

name	The name of this node (for navigation)
type	The type of this node. This type also determines which functions can be applied to this node later on.

3.2.5 ORM_NodeDelete

Deletes the given node and all its children e.g. returns the space allocated Note: if the nodes parent pointer is not NULL, the node will not be deleted.

Prototype:

```
int
ORM_NodeDelete( ORM_NodeDef node    /* in */
              );
```

Parameters:

node	The node (and the subtree) to delete

3.2.6 ORM_NodeAttach

Attaches a node (and its subtree) into an existing tree as a new subtree. Every node (subtree) is in at most 1 tree!

Prototype:

```
int
ORM_NodeAttach( ORM_RelationDef relation,    /* in */
                ORM_NodeDef     relative,    /* in */
                ORM_NodeDef     subtree      /* in */
              );
```

Parameters:

relation	: Flag either ORM_NodeSibbling or ORM_NodeChild, specifying the role of the *relative* node, e.g. its a sibbling or its the parent of the subtree to attach. If its a parent, the new node will be attached at the end of all children, if its a sibbling, it will be placed right before this child.
relative	: an existing node, either parent of sibbling
subtree	No description

3.2.7 ORM_NodeDetach

Detaches a subtree from the current root tree. This always has to be called, before a subtree is actually deallocated. The subtree may also be reattached in the same tree again after this call

Prototype:

```
int
ORM_NodeDetach(  ORM_NodeDef  subtree      /* in */
              );
```

No parameter descriptions are available.

3.2.8 ORM_NodeHandleSet

Sets the handle in the given node (see also ORM_Node<convenience functions>)

Prototype:

```
void
ORM_NodeHandleSet(  ORM_NodeDef          node,      /* in */
                    ORM_AppHandleDef     handle     /* in */
                 );
```

Parameters:

node	Reference to node structure of any type.
handle	Reference to opaque handle.

3.2.9 ORM_NodeHandleGet

Retrieves the handle from a given node

Prototype:

```
int
ORM_NodeHandleGet(  ORM_NodeDef          node,      /* in */
                    ORM_AppHandleDef     *handle    /* out */
                 );
```

No parameter descriptions are available.

3.2.10 ORM_NodeAspectSet

Sets the aspect in the given node (see also ORM_Node<convenience functions>)

Prototype:

```
void
ORM_NodeAspectSet(  ORM_NodeDef          node,      /* in */
                    ORM_AppAspectDef     aspect     /* in */
                    );
```

Parameters:

node	Reference to node structure of any valid node type.
aspect	Reference to opaque aspect description.

3.2.11 ORM_NodeAspectGet

Retrieves the aspect from a given node

Prototype:

```
int
ORM_[illegible]  ORM_NodeDef          node,      /* in */
                 ORM_AppAspectDef     *aspect    /* out */
                 );
```

No parameter descriptions are available.

3.2.12 ORM_NodeAttributeDescrSet

Sets the attribute description of an attribute node

Prototype:

```
int
ORM_[illegible]  ORM_NodeDef              node,      /* in */
                 ORM_AppAttribDescrDef    attrib     /* in */
                 );
```

Parameters:

node	Reference to node structure of type Attribute.
attrib	Reference to opaque attribute description

3.2.13 ORM_NodeAttributeDescrGet

Gets the attribute description of an attribute node

Prototype:

```
int
ORM_NodeAttributeDescrGet(  ORM_NodeDef              node,       /* in */
                            ORM_AppAttribDescrDef    *attrib     /* out */
                            );
```

No parameter descriptions are available.

3.2.14 ORM_NodeObjectLinkSet

Sets the link of an object node

Prototype:

```
int
ORM_NodeObjectLinkSet(  ORM_NodeDef node,          /* in */
                        ORM_String  link           /* in */
                        );
```

Parameters:

node	Reference to node structure of type Object.
link	Stringified version of the address / nor to call this object.

3.2.15 ORM_NodeObjectLinkGet

Gets the linkaddress of an object node

Prototype:

```
int
ORM_NodeObjectLinkGet(  ORM_NodeDef node,          /* in */
                        ORM_String  *link          /* out */
                        );
```

No parameter descriptions are available.

3.2.16 ORM_NodeObjectAdd

for an explanations of paramters, see above. Return created node if operation succeeded else NULL.

Prototype:

```
ORM_NodeDef
ORM_NodeObjectAdd(      ORM_RelationDef     relation,       /* in */
                        ORM_NodeDef         relative,       /* in */
                        ORM_String          name,           /* in */
                        ORM_AppHandleDef    handle,         /* in */
                        ORM_AppAspectDef    aspect,         /* in */
                        ORM_String          linkaddr        /* in */
                    );
```

No parameter descriptions are available.

3.2.17 ORM_NodeComponentAdd

Prototype:

```
ORM_NodeDef
ORM_NodeComponentAdd(   ORM_RelationDef     relation,       /* in */
                        ORM_NodeDef         relative,       /* in */
                        ORM_String          name,           /* in */
                        ORM_AppHandleDef    handle,         /* in */
                        ORM_AppAspectDef    aspect          /* in */
                        );
```

No parameter descriptions are available.

3.2.18 ORM_NodeAttributeAdd

Prototype:

```
ORM_NodeDef
ORM_NodeAttributeAdd(   ORM_RelationDef     relation,       /* in */
                        ORM_NodeDef         relative,       /* in */
                        ORM_String          name,           /* in */
                        ORM_AppHandleDef    handle,         /* in */
                        ORM_AppAspectDef    aspect,         /* in */
                        ORM_AppAttribDescrDef attribdescr   /* in */
                        );
```

No parameter descriptions are available.

3.3 ORM Aspect Layer

This section was generated from *<stdin>* by CDOC on Sun Jan 29 17:00:51 1995.

The ORM aspect layer adds another level of ORM application/server support on top of the ORM Node/Handle layer, and supports the retrieval and modification of aspects, i.e. groups of attributes from or into application data structures, once those have been registered with this layer.

This level has no additional (down-call) functions but defines data structures to be provided by the application layer. These are then accessed/used by the aspect upcall functions, if those have been registered with the ORM protocol layer.

The Aspect layer implementation of the ORM-SSL works as follows:

On AspectCallGet requests, just a pointer is returned which points at offset bytes (as set in the aspect descriptor) from the beginning of the handle. On AspectCallInit calls, a copy of the aspect, e.g. size bytes from the area pointed to by handle, starting from offset, is taken into a private memory area. This copy is then passed to the Attribute conversion routines to write the new values into. On AspectCallSet calls, the application level set function as denoted by the aspect descriptor is called and the private copy (request structure) is released afterward.

3.3.1 Function Type ORM_AspectSetFunc

This function is called from the aspect layer to actually apply the new attribute values to the application layer and/or initiate the requested state changes. This function usually should not block, e.g. should not wait until the initiated state change is completed. Any kind of intermediate state should instead be visible to a client on request (i.e. not STOPPED -> STARTED, but STOPPED -> STARTING -> STARTED, if starting implies a *heavier* operation.

Declaration:

```
ORM_Status
typedef (*ORM_AspectSetFunc)(
    ORM_AppHandleDef          handle,          /* in */
    ORM_AspectAttributeDef    aspect,          /* in */
    ORM_AppDataPtrDef         request,         /* in */
    ORM_AppDataPtrDef         current,         /* in */
    ORM_String                *errortext       /* out */
    );
```

Fields:

handle	the handle as returned from HandleGet
aspect	Reference to the aspectdescr.
request	Copy of the aspect as described by the aspectdescr updated with new values.

current	Reference to aspect within handle
errortext	Where to store a point r to a short textual description if the requested values could NOT be applied.
returns	ORM_ENoError if all new values could be applied, or ORM_EParameterList if paremter set is inconsistent or ORM_EMissingAttribute if a mandatory attribute is NULL.

3.3.2 The ORM_AspectDescrDef

This descriptor maintains information about the application data structure (usually references by the ORM_AppHandle) or parts of it. It describes the binary size, the offset within the handle, and contains pointers to functions to actually retrieve or modify this aspect of the application instance.

Note: It is currently open, whether there should be a procedural interface to set up the aspect descriptor instead of providing a structure type definition to be passed initialized by the application code.

Declaration:

```
typedef struct _ORM_AspectDescrTag {
        char                    *name;
        size_t                  offset;
        size_t                  size;
        long                    flag;
        ORM_AspectSetFunc       setf;
        long                    appid;
        void                    *appext;
        } ORM_AspectDescrDef;
```

Fields:

name	Pointer to name string, for identification mainly.
offset	The offset in bytes within the instance, where this aspect starts. This usually is the offset of a sub structure in the instance.
size	The size in bytes of the instance, the application handle pointer points to. For set-requests, the container for the new value is created by copying the handle, and inserting the new values in it.
flag	If set to ORM_AspectGetIndirect, the offset indicates the offset to a pointer, pointing to another structure of the above size.
setf	Pointer to function, which is called to apply (a set of) new values to an application instance.
appext	any value of pointer size the application wants to store with the aspect. This may be used to store a create_aspect function pointer.
appid	Opaque identifier, which may be used by the applications layer

3.4 ORM Attribute Layer

This section was generated from *<stdin>* by CDOC on Fri Jan 27 19:59:34 1995.

The ORM Attribute layer adds another level of ORM application/server support on top of the ORM node layer, by providing (list of) attribute descriptors, which simply initialized by the application code, allowe automatic conversion and generation of the attribute meta information, requested by the ORM protocol layer.

The implementation of the attribute layer in the ORM SSL assumes, that it is converting to and from a binary blob of data, identified by the (lower level) aspect descriptor. The goal of this layer is to reduce the coding effort needed by the application writer at this layer, just to provide some initialized descriptors and pass them to the ORM SSL via single calls per every instance created.

3.4.1 The ORM_AttributeDescriptorDef

This data structure describes a single attribute, e.g. its native type and mode, its size, pointers to conversion functions. In addition it maintains hooks for preset meta-info like - *Unit* and *Range*.

Declaration:

```
typedef struct ORM_AttributeDescrTag {
    ORM_String                          name;
    ORM_AttribTypeDef                   datatype;
    ORM_AttribModeDef                   accessmode;
    ORM_String                          range;
    ORM_String                          unit;
    size_t                              offset;
    size_t                              size;
    ORM_ConverterNativeToStringFunc     nativetostring;
    ORM_ConverterStringToNativeFunc     stringtonative;
    ORM_AppConverterArgDef              convarg;
} ORM_AttributeDescrDef;
```

Fields:

name	The name of the attribute.
datatype	The type of data of this attribute (ORM_AttributeTypeDef). This is a superset of the data types, the ORM protocol defines and used to determine implicit conversion routines.
mode	The allowed access modes of this attribute out of ORM_AttribMode values, e.g. read-only, write- only, read-write.
range	A string describing the allowed ranges for new values for read-write or write-only attributes only. This is a *ORM hint*, and as such optional
unit	A unit string (usually *ms*, *Mb*, etc.) which may be used by object specific user interface generators in any way, and by default if

present is placed behind the attribute value. This is also an *ORM hint* and as such optional.

conversion function A function pointer to an application specific conversion function, to convert between native and ORM presentations. *Note:* This is not to be confused with the similar functions of the ORM_Context structure. For the <conversion-function> to be called, the ORM_Node.conversions functions have to be setup in the ORM_Context.

conversion-arg An opaque pointer to any argument, the conversion function may need to convert this attribute.

3.4.2 ORM_AttributeCreate

This function combines several actions required to register an attribute of a (new) instance with the ORM SSL, i.e. it creates an attribute node under the given parent (which must be of ORM_NodeTypeComponent) and attaches the attribute description and the handle information to it.

Prototype:

```
int
ORM_AttributeCreate(
            ORM_NodeDef                 relative,    /* in */
            ORM_RelationDef             relation,    /* in */
            ORM_AttributeDescrListDef   attribdescr, /* in */
            ORM_AspectDescrListDef      aspectdescr, /* in */
            ORM_AppHandleDef            handle,      /* in */
            ORM_NodeDef                 *new         /* out */
            );
```

Parameters:

relative pointer to relative node. If *relation* is set ORM_NodeIsParent, then this has to be a node of ORM_NodeTypeComponent. If *relation* is set to ORM_NodeIsSibbling, then this node can be of any valid node type.

relation Either ORM_NodeIsParent, if the node *relative* should be the parent of the new attribute node, or ORM_NodeIsSibbling, if the new attribute node should be inserted after the *relative* node as a sibbling.

attribdescr No description

aspectdescr No description

handle Pointer to the application instance this attribute belongs to or ORM_HandleInherit (-1), if the handle should be taken from the parent (or its parent and so on).

new Pointer to new attribute node or NULL on failure.

3.4.3 ORM_AttributeDestroy

This function detaches the attribute node from the tree of nodes if any, deletes the node structure and deletes any depending structures, i.e. the attribute descriptor.

In the current implementation this function maps directly to ORM_NodeDestroy, but nevertheless this function should be called for attribute nodes created with functions of this layer to be able to deallocate any dynamic memory.

Prototype:

```
int
ORM_AttributeDestroy(    ORM_NodeDef attrnode);
```

Parameters:

attrnode	Pointer to attrbute node.

3.4.4 ORM_AttributeListCreate

This is another convenience functions to add a list of attributes to a component. The given node must be a of component type and is used as the parent for the new list of attributes (which is appended to the end of the list of child-nodes). The pointer to the attribute descriptor now points to an array of those descriptors, where the end of the array is marked by a descriptor whose name pointer is NULL.

Prototype:

```
int
ORM_AttributeListCreate(
        ORM_NodeDef                 parent,        /* in */
        ORM_AppHandleDef            handle,        /* in */
        ORM_AspectDescrDef          aspectdescr,   /* in */
        ORM_AttributeDescrListDef   attrdescrlist, /* in */
        long                        attrcount      /* in */
        );
```

Parameters:

parent	Pointer to an existing component node, who is the parent node of all newly created attribute nodes.
handle	Pointer to the application instance, all attribute belongs to or ORM_HandleInherit (-1), which indicates, that the actual handle is determined by the parent (which again may have its handle set to ORM_HandleInherit!)
aspectdescr	No description
attrdescrlist	Pointer to an array of ORM_AttributeDescr, with name=NULL in the last element if *attrcount* is < 0.

attrcount	The number of attribute descriptors in the list or the number of initial attributes from this list to attach to this node or *-1*, if the end of the list (array) should be determined by a NULL nodeinfo pointer.

3.5 ORM Attribute Conversion Support

This section was generated from *<stdin>* by CDOC on Sun Jan 29 18:13:38 1995.

This part or the ORM Server Support Layer provides functions for converting generic ORM data types between their native (binary) and the ORM (ASCII) presentation. The interface between the attribute and the conversion layer is defined by to function types, one for converting application native data into an ORM representation, one to convert ORM attribute value strings into the applications native presentation. Beside the conversion functions provided by the ORM-SSL, every application may provide its own special converters as long as their interfaces conform these function types.

3.5.1 Function Type ORM_ConverterNativeToString

This function is called to convert a single native value into its string representation. In addition to the value string it may generate the range and unit strings, if the pointer values passed are non-null. If the converter function returns NULL in these pointers, the lower (attribute) layer may provide default strings if any.

Memory Allocation: The memory to hold the converted string value(s) has to be provided by the converter function. It is reasonable to use static memory for this purpose, because before the converter function is called again, the ORM protocol layer will copy the strings returned.

Declaration:

```
typedef ORM_Status (* ORM_ConverterNativeToStringFunc)(
            ORM_AppDataPtrDef          ptr,          /* in */
            size_t                     size,         /* in */
            ORM_AttributeDescrListDef  datatype,     /* in */
            ORM_AppConverterArgDef     convarg,      /* in */
            ORM_String                 *strvalue,    /* out */
            ORM_String                 *strrange,    /* out */
            ORM_String                 *strunit      /* out */
            );
```

Fields:

ptr	Address of native data element (e.g. attribute value)
size	Byte-size of data element
datatype	One of the ORM_AttributeTypes identifying the type of the native data element and its mapping to an ORM Protocol data type (??is this overloaded ??)
convarg	Any kind of argument (pointer) for this converter (as provided with the attribute descriptor for ex.)
strvalue	Where to store the pointer to the converted value string.
strrange	Where to store the reference to the optional range string.

strunit Where to store the reference to the optional unit string.

3.5.2 Function Type ORM_ConverterStringToNative

This function is called to convert a single ORM string value into its native presentation. The pointer for the result usually points into a set of different attributes, e.g. an aspect, which usually is a (partial) copy of some application data instance.

Memory Allocation: The destination pointer provided references some valid memory (e.g. an aspect), but for references (the native value is a C-string for ex.), there is usually not enough space for the referenced value. This space must be allocated/provided by the converter itself. It is legal, to reference the original string as passed in to the converter function, but then the AspectCallSet function should make a copy, if the string is needed beyond this call.

Declaration:

```
typedef ORM_Status (* ORM_ConverterStringToNativeFunc)(
            ORM_AppDataPtrDef       dest,           /* in */
            size_t                  size,           /* in */
            ORM_AttributeTypeDef    datatype,       /* in */
            ORM_AppConverterArgDef  converg,        /* in */
            ORM_String              strvalue        /* in */
          );
```

Fields:

dest	Address/destination of native data element (e.g: attribute value)
maxsize	Maximum byte-size of data element
datatype	One of the ORM_AttributeTypes identifying the type of the native data element
converg	Any kind of data (pointer) for this converter as provided with the attribute descriptor
strvalue	The new attribute value in its ascii presentation.
returns	ORM_ENoError if conversion was successfull and the resulting attribute value is valid or ORM_ERangeError.

3.5.3 ORM Built In Conversion Functions

The following functions are provided to convert generic C datatypes between their ORM and their native presentation. In addition sub functions are provided to support the special ORM SELECT and MCHOICE types, which are called by the generic converters. Along with these sets two new data structure (types) are introduced.

3.5.4 Functi n ORM_GenericNativeToString

This function converts standard C-data types into their ASCII presentation. It returns only the converted value, but does not support the range and unit parts (e.g. returns NULL for those, if requested). In case of SELECT or MCHOICE functions, this function calls the related ORM_Select.. or ORM_MChoice functions.

Note: It is currently open, whether the conversion argument *convarg* may be used to specify a format string a la *printf*. Furthermore it is currently open, whether a NULL conversion function in the attribute descriptor should be directed to this (default) function.

Arguments as for ORM_ConverterNativeToString!

Prototype:

```
ORM_Status ORM_GenericNativeToString(
        ORM_AppDataPtrDef       ptr,            /* in */
        size_t                  maxsize,        /* in */
        ORM_AttribTypeDef       type,           /* in */
        ORM_AppConverterArgDef  convarg,        /* in */
        ORM_String              *strvalue,      /* out */
        ORM_String              *rangevalue,    /* out */
        ORM_String              *strunit        /* out */
        );
```

No parameter descriptions are available.

3.5.5 Function ORM_GenericStringToNative

This function converts ASCII C-strings into standard C-datatypes. In case of SELECT or MCHOICE functions, this function calls the related ORM_Select.. or ORM_MChoice functions.

Note: It is currently open, whether the conversion argument *convarg* may be used to specify a format string a la *sscanf*. Furthermore it is currently open, whether a NULL conversion function in the attribute descriptor should be directed to this (default) function.

Arguments as for ORM_ConverterStringToNative!

Prototype:

```
ORM_Status ORM_GenericStringToNative(
        ORM_AppDataPtrDef       ptr,            /* in */
        size_t                  maxsize,        /* in */
        ORM_AttribTypeDef       type,           /* in */
        ORM_AppConverterArgDef  convarg,        /* in */
        ORM_String              strvalue        /* in */
        );
```

No parameter descriptions are available.

3.5.6 Structure ORM_StringMapDef

This type of structure is used to map strings to binary values and vice versa. It may be used to convert internal flags and states to *friendly* names. StringMaps must be terminated by an entry with *name* set to NULL.

Declaration:

```
typedef struct ORM_StringMapTag {
    ORM_String  name;
    ORM_Key     key;
} ORM_StringMapDef;
```

Fields:

name	Friendly name for this key.
key	The binary native value of the key

3.5.7 ORM_StringMapToString

This function maps a value key to a string using the given StringMap. It returns the string of that map entry, whose key is equal to the given key, else it returns the string passed in *notfound*.

Prototype:

```
ORM_String
ORM_StringMapToString( ORM_StringMapDef    map,
                       ORM_Key             key,
                       ORM_String          notfound);
```

Parameters.

map	Pointer to a sequence of map entries
key	Binary key value.
notfound	string to give back, if none of the keys in the map matched.

3.5.8 ORM_StringMapToKey

This function maps a string value to a binary key using the given StringMap. It returns the key of that map entry, whose string is equal to the given key, else it returns the key passed in *invalidkey*.

Prototype:

```
ORM_Key
ORM_StringMapToKey( ORM_StringMapDef    map,
                    ORM_String          name,
                    ORM_Key             invalidkey);
```

Parameters:

map	Pointer to a sequence of map entries
name	No description
invalidkey	No description

3.5.9 Structure ORM_StateMapDef

This structure is used to map *states* into strings, where a *state* is assumed to have a distinct set of possible next states, depending on the current value. E.g. this structure can be used to derive the set of possible new values i.e. it can provide the *range* value for a state attribute.

Otherwise it is used similar to the simpler StringMap structure. StateMaps must be terminated by an entry with *name* set to NULL.

Declaration:

```
typedef struct _ORM_StateMapTag { /* not of the U.S.A.!! */
    ORM_String  name;
    ORM_Key     state;
    ORM_String  validnexts;
} *ORM_StateMapDef;
```

Fields:

name	Friendly name for this key.
state	The binary native value of this state
validnexts	String of comma separated names of next valid states which may follow this state.

3.5.10 ORM_StateMapToString

Convert an encoding of a state into a *friendly* name using the given statemap. If the state could not be found, the string passed in *notfound* is returned.

Prototype:

```
ORM_String
ORM_StateMapToString (  ORM_StateMapDef map,
                        ORM_Key     state,
                        ORM_String  notfound);
```

Parameters:

map	Pointer to a (name=NULL) terminated state map.
state	the binary state
notfound	string to return, if none of the entries in the map had exactly the given state key.

3.5.11 ORM_StateMapToKey

Convert a string representation of a state into a native encoding of a *state using the given statemap*. If the string could not be found, the state passed in *invalidstate* is returned.

Prototype:

```
ORM_Key
ORM_StateMapToKey ( ORM_StateMapDef map,
                    ORM_String  name,
                    ORM_Key     invalidstate);
```

Parameters:

map	Pointer to a (name=NULL) terminated state map.
name	No description
invalidstate	No description

3.5.12 ORM_StateMapNextByKey

Return the comma separated list of valid next states given the current state.

Prototype:

```
ORM_String
ORM_StateMapNextByKey(  ORM_StateMapDef map,
                        ORM_String  state);
```

Parameters:

map	Pointer to a (name=NULL) terminated state map.
state	the binary state

3.6 ORM Dump & Restore Support

This section was generated from *<stdin>* by CDOC on Fri Jan 27 19:59:34 1995.

This module of the ORM Server Support Library supports the dump and restore of complete subtrees, and therefore can be used to save the current configuration to a persistant storage media (i.e. the MSF Warehouse) and reload it from there. The actual IO functions are currently not supported by this layer or the support library at all!

Dump and Restore are functions of the ORM SSL and not of the ORM protocol (i.e. there is no *DUMP* or *RESTORE* request defined in the protocol).

This implies, that these functions have to be dispatched out of the application layer explicitly. One (intended) way to dispatch those functions interactively is to provide pseudo components in every subtree, which should be independent storable/reloadable. These contain the required parameters like Warehouse location or version name as attributes. An Attribute-Set request to this subtree then results in the execution of the corresponding function.

Under the layered view of the ORM SSL, these two functions belong to the protocol layer, as they use (nearly) the same functionality of the higher layers via upcalls.

3.6.1 General Model:

Starting from a given node, which is used as the root of the relevant subtree to dump, all components, object links and writable attributes with their meta information are recursivly extracted relative to the current subtree root. The extended/meta information on the persistent media can be used to interprete the stored attributes and apply changes to the stored version without the ORM server/application alive but through special clients (by an ORM/Warehouse gateway for example).

The dumped ORM tree can be used to reload the whole subtree at any time, by providing the node and call the *restore* function of the ORM SSL (which is a special kind of Set-Request).

This special kind of SET request creates a new situation, as components (or any new subtree) may have been created dynamically by the ORM server application on request. On the next cold start of the application, these subtrees do not exist.

This results in failed lookup requests by the ORM protocol layer, which usually is treated as an error (remember: ORM-P has no direct support for object/component creation, but this is emulated by sets of writeonly attributes in separate subtrees, i.e. *New..*). To handle this case, the application can provide a special function during application context setup to create new instances including the ORM subtree (ORM_NodeNotFoundTrapFunc()).

A parameter is passed to this creation function, which indicates, whether this situation was caused by a regular ORM protocol request or by an internally generated *restore* request, so the application code can still decide to refuse the creation.

3.6.2 ORM_Dump

This function extracts the ORM entities in the subtree pointed to by *subtree* into the character buffer, so it can be used by a later ORM_Restore function (or can be used as a subrequest in a regular ORM protocol request).

It is the responsibility of the caller to provide a sufficient buffer, which can hold the subtree information of the given depth!

Prototype:

```
ORM_Status
ORM_Dump(       ORM_AppNodeRef  subtree,
                long            depth,
                long            what,
                ORM_String      buffer,
                long            *maxlen
                );
```

Parameters:

subtree	The root of the subtree to dump. All navigation information is saved relative to this node.
depth	The depth, up to which entities in this subtree should be extracted. A depth of *0* means, direct childs of the given sub-root only, i.e. if the subtree points to a component node with an attribute node as one of its direct children, the name of the attribute would be extracted, but not the value or other extended attribute information, if depth=0. A depth of *-1* extracts the whole subtree, independent of its depth.
what	Is a bitmask, defining what kind of entities should be extracted: ORM_DumpSetObjects ORM_DumpSetComponents ORM_DumpSetAttributes ORM_DumpSetWritable ORM_DumpSetDefault = Objects | Components | Writable ORM_DumpSetEveryThing = Objects | Components | Attributes
buffer	The address of a character buffer, where to store the extracted entity information
maxlen	Pointer to the maximum length of this buffer. On return, maxlen will contain the number of bytes used in this buffer including the C-String '\0' terminator.

3.6.3 ORM_Restore

Function to reload the saved ORM information into an existing subtree, where at least the root of the given subtree has to exist. *Note:* Because the restore request may fail with som attribute modifications already performed, an application may want to call ORM_Dump

(int a t mporary buffer) before actually calling ORM_Restore, to be able t *undo* the partial operations.

Prototype:

```
ORM_Status
ORM_Restore(    ORM_ApplSubTree  subtree,
                ORM_String       buffer,
                long             *length
                );
```

Parameters:

subtree	Node of the subtree to load the management information into.
buffer	pointer to ORM subrequest sequence.
length	pointer to length of the request. On return, this will contain the number of bytes processed from this request.

3.7 ORM SSL Generic Datatypes

```
#ifndef _ORM_TYPE_H
#define _ORM_TYPE_H

/*
 * Some generic definitions, may become obsolete
 */

typedef enum {
    False,
    True
    } boolean;

#ifndef NULL
#define NULL (void *)0
#endif

typedef unsigned long size_t;

#define ORM_Ptr( base, offset) (void *)((size_t )(base)+(size_t)(offset))

#define ORM_Malloc(x)    (void *)malloc(x)
#define ORM_Free(x)      free(x)

/*
 * more ORM specific stuff
 */

/*
 * How requests and responses are passed to the ORM protocol layer
 */

typedef char     *ORM_RequestDef;

typedef char     *ORM_ResponseDef;

/*
 * The principal type of every ORM protocol entity, e.g. names and values,
 * but also used most C-strings.
 */

typedef char     *ORM_String;

/*
 * Used for StateMaps and String Maps as the lookup key
 */

typedef long     ORM_Key;


/*
 * The following are various opaque handles. Opaque mainly to the protocol
 * layer but also for the lower of two stacked layers.
 */

typedef void     *ORM_AppNodeDef;
typedef void     *ORM_AppHandleDef;
typedef void     *ORM_AppAspectDef;
typedef void     *ORM_AppDataPtrDef;
```

```
typedef void     *ORM_AppAttribDescrDef;
typedef void     *ORM_AppCallContextDef;
typedef void     *ORM_AppConverterArgDef;

/*
 * Valid Access modes for an attribute
 */

typedef enum {
    ORM_AttribModeNone,
    ORM_AttribModeRW,
    ORM_AttribModeRO,
    ORM_AttribModeWO,
    ORM_AttribModeRWP
    } ORM_AttribModeDef;

/*
 * Known (native) datatypes, which are supported by the Generic converter
 */

typedef enum {
    ORM_AttribTypeNone,
    ORM_AttribTypeInt1,
    ORM_AttribTypeUInt1,
    ORM_AttribTypeInt2,
    ORM_AttribTypeUInt2,
    ORM_AttribTypeInt4,
    ORM_AttribTypeUInt4,
    ORM_AttribTypeInt8,
    ORM_AttribTypeUInt8,
    ORM_AttribTypeReal32,
    ORM_AttribTypeReal64,
    ORM_AttribTypeString,
    ORM_AttribTypeHexDot,
    ORM_AttribTypeSelect,     /* 1 out of many */
    ORM_AttribTypeState,      /* 1 out of many, but with dynamic range */
    ORM_AttribTypeOption,     /* binary switch ON/OFF YES/NO */
    ORM_AttribTypeMChoice,    /* n out of many */
    ORM_AttribTypeUnknown
    } ORM_AttribTypeDef;

/*
 * ORM Error Codes, used as well by the protocol as by the ORM SSL
 */

typedef enum {
    ORM_ENoError,             /* Operation successfull! */
    ORM_EPermission,          /* None or wrong auth.information */
    ORM_ENoSuchNode,          /* some name in pathname could not be found */
    ORM_ENoSuchAttribute,     /* Attribute in Set-Request doesn't exist */
    ORM_ENoSuchObject,        /* Object/Manager could not be found */
    ORM_EInvalidOperation,    /* Operation not applicable to node type */
    ORM_EProtocol,            /* ORM protocol violation */
    ORM_ECommunication,       /* lower level comm error */
    ORM_ERange,               /* new attribute value out of range */
    ORM_EParameterList,       /* set of attributes not applicable */
    ORM_EMissingAttribute,    /* mandatory attrib. missing or NULL */
    ORM_ENoSpace,             /* internal allocation */
    ORM_ENoBuffer,            /* response buffer */
```

```
    ORM_EInternal,              /* ORM Internal error -> bug */
    ORM_EApplication,           /* application level error -> bug */
    | ORM_Status;


/*
 * Types of nodes in the virtual tree. Note, that only nodes of type
 * ORM_NodeTypeComponent can have children!
 */

typedef enum {
    ORM_NodeTypeUnknown,
    ORM_NodeTypeObject,
    ORM_NodeTypeComponent,
    ORM_NodeTypeAttribute,
    ORM_NodeTypeAny
    | ORM_NodeTypeDef;

/*
 * Types of ORM requests. Note that the Dump and Restore requests are
 * not part of the ORM protocol, but only available within the ORM
 * server support library
 */

typedef enum {
    ORM_RequestObjectGet,
    ORM_RequestComponentGet,
    ORM_RequestAttributeGet,
    ORM_RequestAttributeInfoGet,
    ORM_RequestAttributeSet,
    ORM_RequestSet,
    ORM_RequestGet,
    ORM_RequestDump,
    ORM_RequestRestore
    | ORM_RequestTypeDef;


#endif
```

WHAT IS CLAIMED IS:

1. A system for managing objects, including a first server, comprising:

a first receiver portion configured to receive a request in a hypermedia format;

a first translator portion configured to convert the hypermedia request to an object request;

a sender portion configured to send the object request to an object manager;

a second receiver portion configured to receive a response from the object manager; and

a second translator portion configured to convert the object manager response to the hypermedia format.

2. The system of claim 1, further comprising a second server, including:

a third receiver portion configured to receive a request in a hypermedia format;

a third translator portion configured to convert the hypermedia request to an object request;

a second sender portion configured to send the object request to an object manager;

a fourth receiver portion configured to receive a response from the object manager; and

a fourth translator portion configured to convert the object manager response to the hypermedia format.

3. The system of claim 1, further comprising:

a second sending portion configured to send the hypermedia format data from the sender portion to a browser to be displayed.

4. The system of claim 1, where the object manager manages a self-describing object.

5. The system of claim 1, where the object manager manages a non-self describing object.

6. The system of claim 5, where the object manager performs a "worm" function.

7. A method for browsing objects, where a browser communicates with a server, comprising the steps, performed by the browser, of:

sending an initial URL to the server;

receiving first data from the server, where the first data specifies an object corresponding to the URL;

sending user-entered data associated with the object to the server; and

receiving second data from the server, where the second data specifies a second object corresponding to the user-entered data.

8. The method of claim 7,

wherein the step of sending an initial URL to the server comprises the step of sending an initial URL known to the browser, where the URL is the URL of the server.

9. The method of claim 7,

wherein the step of sending an initial URL to the server comprises the step of sending an initial URL entered by the user, where the URL is the URL of the server.

10. The method of claim 7,

wherein the step of sending user-entered data associated with the object to the server includes the step of indicating a "set" operation in the user-entered data.

11. The method of claim 7,

wherein the step of sending user-entered data associated with the object to the server includes the step of indicating a "get" operation in the user-entered data.

12. The method of claim 7, wherein the step of receiving second data from the server includes the step of receiving data corresponding to an attribute value of the object.

13. The method of claim 7, wherein the step of receiving second data from the server includes the step of receiving data corresponding to a second object linked to the first object via an object-link.

14. A computer program product comprising:

a computer usable medium having computer readable code embodied therein for managing objects, the computer program product comprising:

computer readable program code devices configured to cause a computer to effect receiving a request in a hypermedia format;

computer readable program code devices configured to cause a computer to effect converting the hypermedia request to an object request;

computer readable program code devices configured to cause a computer to effect sending the object request to an object manager;

computer readable program code devices configured to cause a computer to effect receiving a response from the object manager; and

computer readable program code devices configured to cause a computer to effect converting the object manager response to a second hypermedia format.

15. The computer program product of claim 14, further comprising:

computer readable program code devices configured to cause a computer to effect sending the second hypermedia format data to a browser to be displayed.

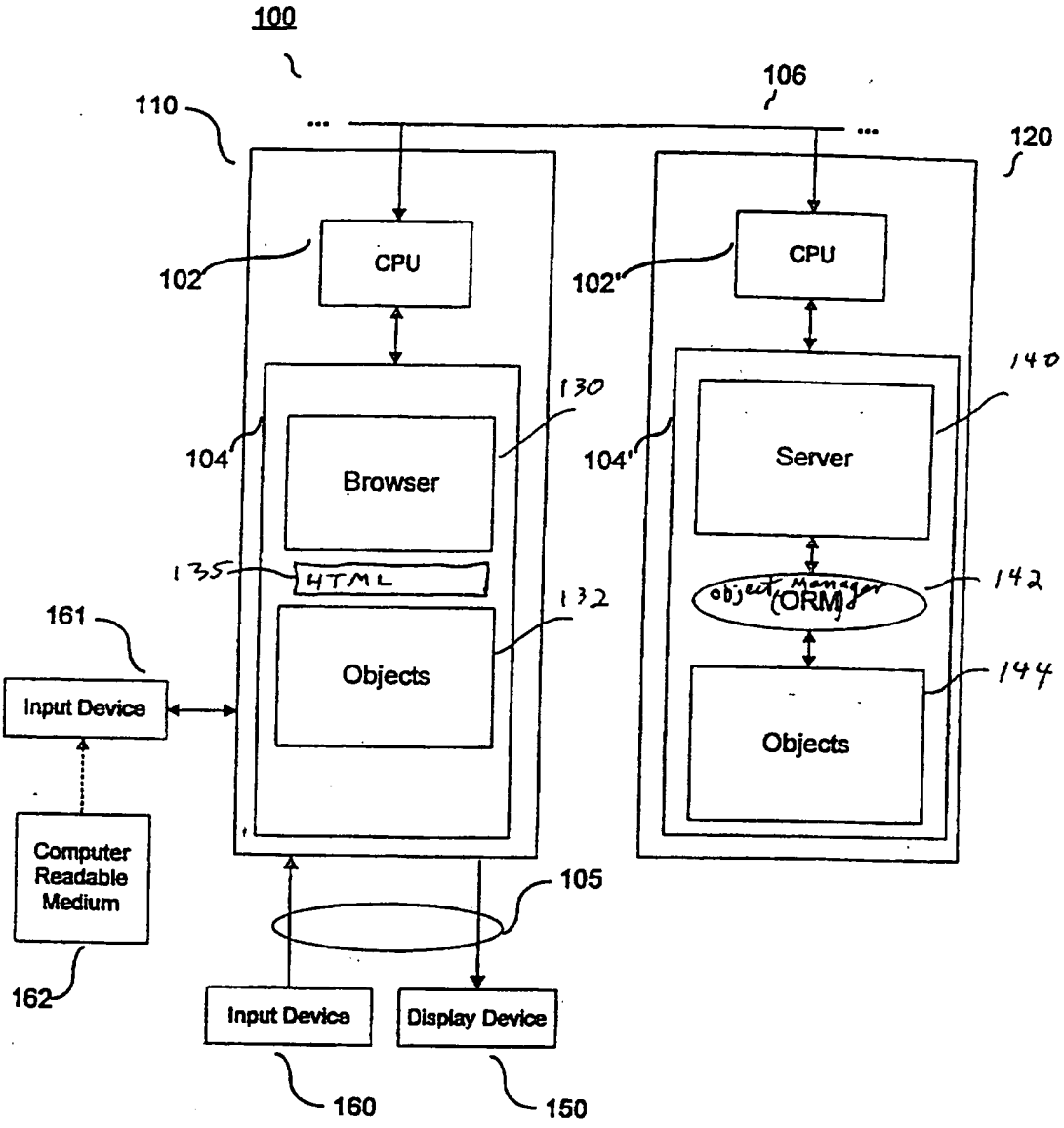

100
106
110
120
CPU
102
102'
CPU
130
140
104
Browser
104'
Server
135
HTML
142
object Manager (ORM)
132
161
Objects
144
Input Device
Objects
Computer Readable Medium
105
162
Input Device
Display Device
160
150

Fig. 1

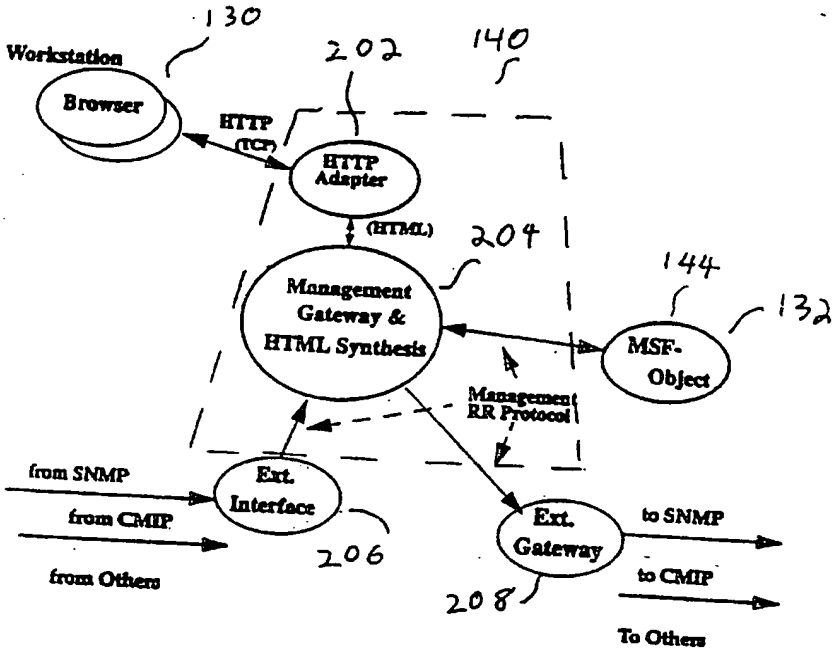

130
Workstation
Browser
202
140
HTTP
(TCP)
HTTP
Adapter
(HTML)
204
144
132
Management
Gateway &
HTML Synthesis
MSF-
Object
Management
RR Protocol
from SNMP
from CMIP
from Others
Ext.
Interface
206
Ext.
Gateway
208
to SNMP
to CMIP
To Others

Fig. 2

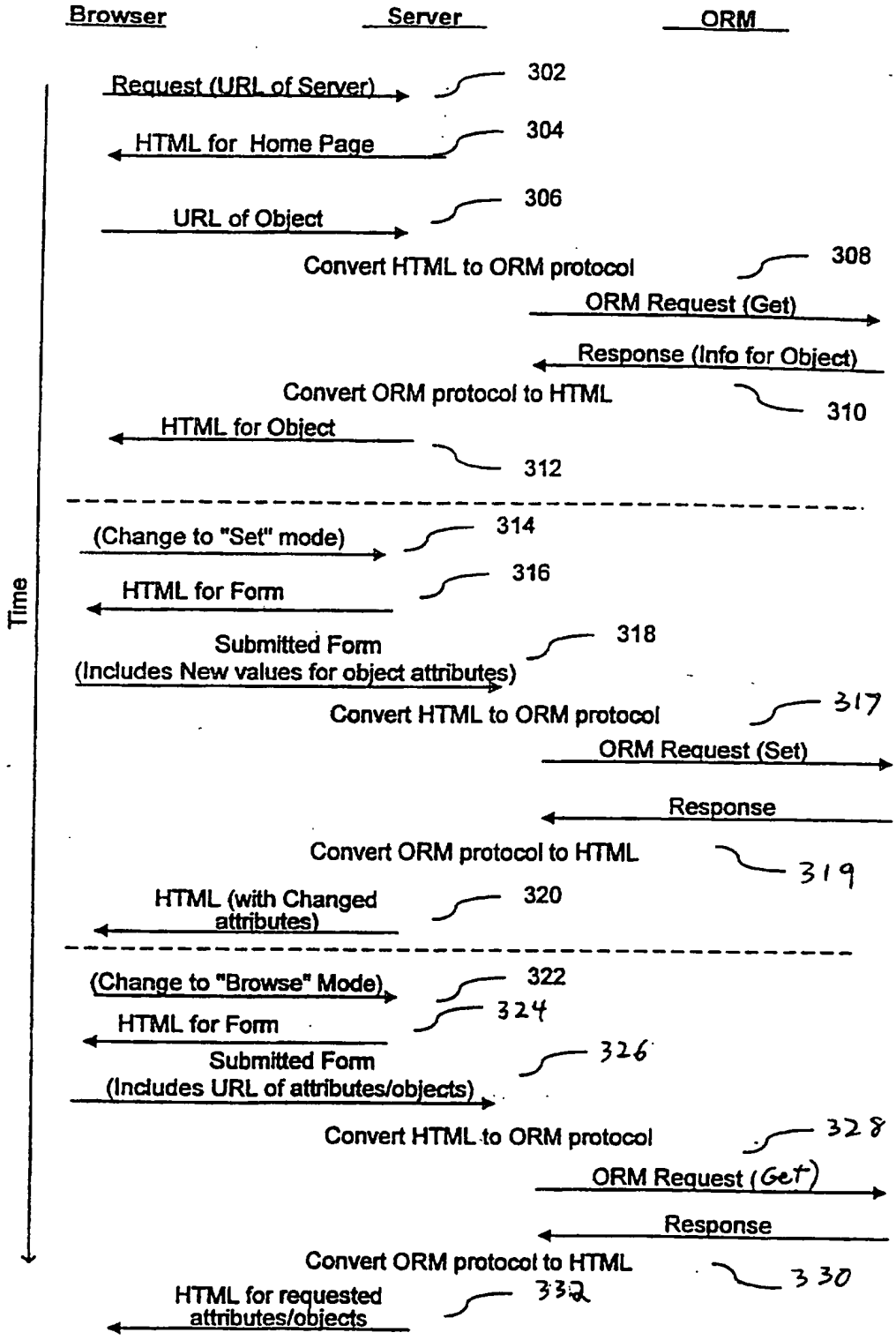

Browser
Server
ORM
Request (URL of Server)
302
HTML for Home Page
304
URL of Object
306
Convert HTML to ORM protocol
308
ORM Request (Get)
Response (Info for Object)
Convert ORM protocol to HTML
310
HTML for Object
312
(Change to "Set" mode)
314
HTML for Form
316
Submitted Form
(Includes New values for object attributes)
318
Convert HTML to ORM protocol
317
ORM Request (Set)
Response
Convert ORM protocol to HTML
319
HTML (with Changed attributes)
320
(Change to "Browse" Mode)
322
HTML for Form
324
Submitted Form
(Includes URL of attributes/objects)
326
Convert HTML to ORM protocol
328
ORM Request (Get)
Response
Convert ORM protocol to HTML
330
HTML for requested attributes/objects
332
Time

Fig. 3

A System as a Tree of Managed Entities:

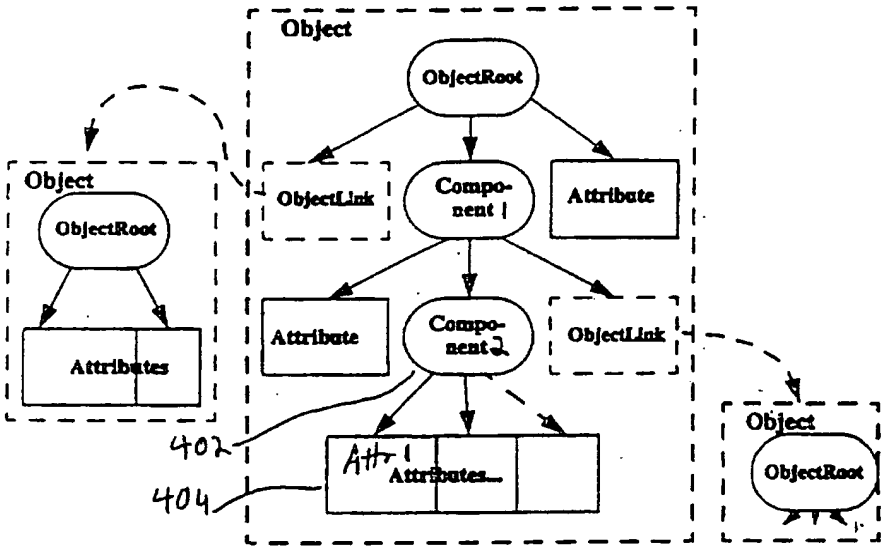


Fig. 4

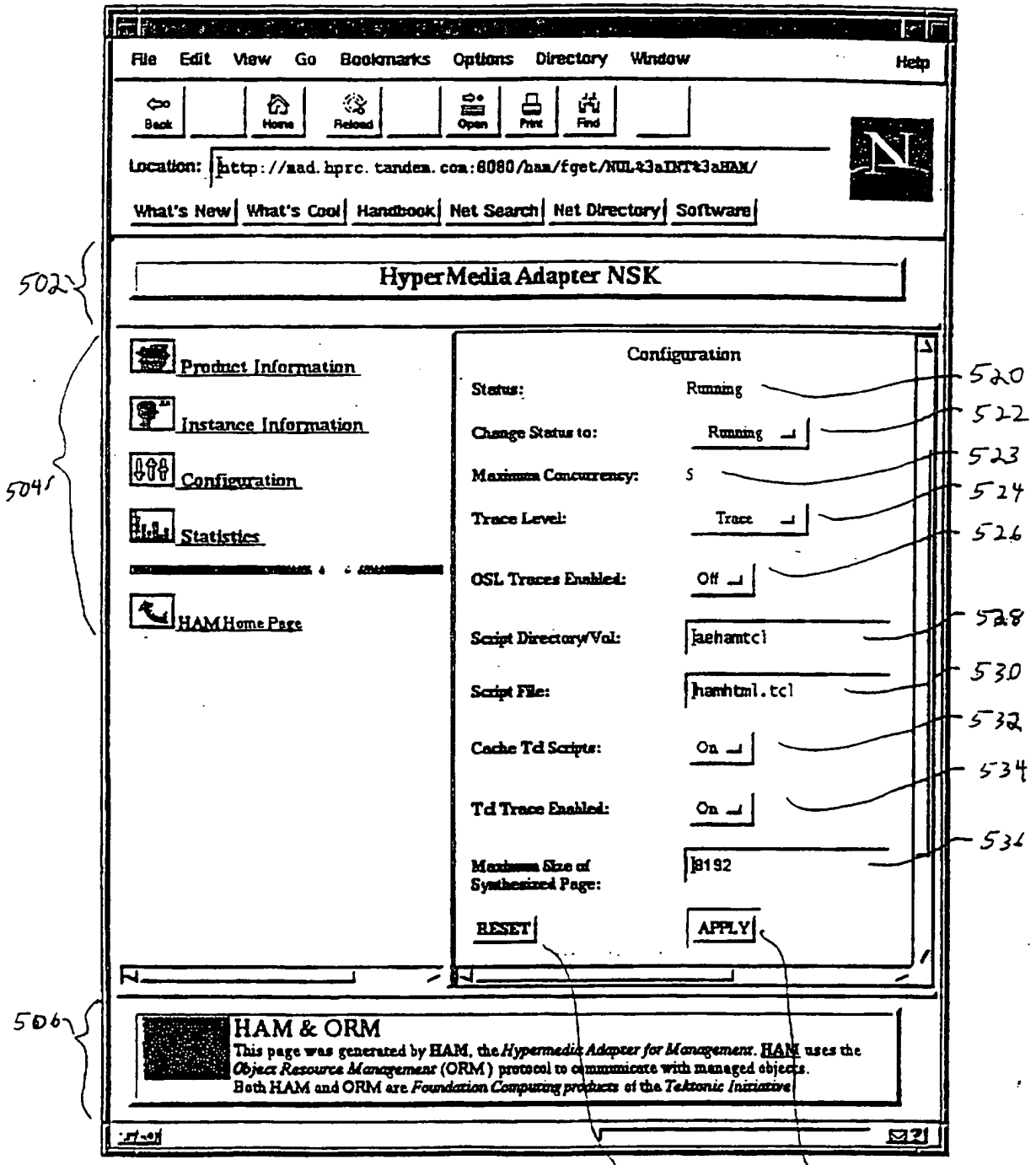

File Edit View Go Bookmarks Options Directory Window Help
Back Home Reload Open Print Find
Location: http://nad.hprc.tandem.com:8080/ham/fget/NUL%3aINT%3aHAM/
What's New What's Cool Handbook Net Search Net Directory Software
HyperMedia Adapter NSK
Product Information
Instance Information
Configuration
Statistics
HAM Home Page
Configuration
Status: Running
Change Status to: Running
Maximum Concurrency: 5
Trace Level: Trace
OSL Traces Enabled: Off
Script Directory/Vol: $ehamtcl
Script File: hamhtml.tcl
Cache Tcl Scripts: On
Tcl Trace Enabled: On
Maximum Size of Synthesized Page: 8192
RESET
APPLY
HAM & ORM
This page was generated by HAM, the Hypermedia Adapter for Management. HAM uses the Object Resource Management (ORM) protocol to communicate with managed objects.
Both HAM and ORM are Foundation Computing products of the Tektonic Initiative
502
504
506
520
522
523
524
526
528
530
532
534
536
540
550

Fig. 5

hamsetup.html

```
1  <HTML>
2  <HEAD>
3  <TITLE>Alter Configuration of HyperMedia Adapter NSK@TCP:168.87.28.7/8062:HAM </TI
   +TLE>
4  <BASE HREF="http://mad.hprc.tandem.com:8080/ham/cget/TCP%3a168.87.28.7%2f8062%3aHAM
   +%2fHyperMedia%20Adapter%20NSK%2fConfiguration/">
5  </HEAD>
6
7  <BODY BACKGROUND="/icons/icstgry.gif">
8  <H2>
9  <IMG ALT="" SRC="/icons/pccnfig.gif">
10 Alter Configuration</H2>
11 <H3>of HyperMedia Adapter NSK</H3>
12 <P><CENTER><IMG SRC="/icons/pcbrblu.gif"></CENTER><P>
13 <FORM METHOD=POST ACTION="http://mad.hprc.tandem.com:8080/ham/cset/TCP%3a168.87.28
   +.7%2f8062%3aHAM%2fHyperMedia%20Adapter%20NSK%2fConfiguration/">
14 <TABLE >
15 <P><CAPTION><STRONG><FONT SIZE=4>Configuration</FONT></STRONG></CAPTION>
16 <TR><TH ALIGN=LEFT><P>Status:</TH>
17 <TD ALIGN=LEFT>Running</TD>
18 <TR VALIGN=BASELINE>
19 <TH ALIGN="LEFT"><P>Change Status to:</TH><TD >
20 <SELECT NAME="Change Status to">
21 <OPTION SELECTED> Running
22 <OPTION> Suspended
23 <OPTION> Aborted
24 <OPTION> Stopped
25 </SELECT>
26 </TD></TR>
27 <TR><TH ALIGN=LEFT><P>Maximum Concurrency:</TH>
28 <TD ALIGN=LEFT>5</TD>
29 <TR VALIGN=BASELINE>
30 <TH ALIGN="LEFT"><P>Trace Level:</TH><TD >
31 <SELECT NAME="Trace Level">
32 <OPTION SELECTED> Warnings
33 <OPTION> Errors
34 <OPTION> Info
35 <OPTION> Debug
36 <OPTION> Trace
37 <OPTION> Everything
38 </SELECT>
39 </TD></TR>
40 <TR VALIGN=BASELINE>
41 <TH ALIGN="LEFT"><P>OSL Traces Enabled:</TH><TD >
42 <SELECT NAME="OSL Traces Enabled">
43 <OPTION SELECTED> Off
44 <OPTION> On
45 </SELECT>
46 </TD></TR>
47 <TR VALIGN=BASELINE>
48 <TH ALIGN="LEFT"><P>Script Directory/Vol:</TH><TD >
49 <INPUT NAME="Script Directory/Vol"
50 TYPE=TEXT VALUE="aehamtcl">
51 </TD></TR>
52 <TR VALIGN=BASELINE>
53 <TH ALIGN="LEFT"><P>Script File:</TH><TD >
54 <INPUT NAME="Script File"
55 TYPE=TEXT VALUE="hamhtml.tcl">
56 </TD></TR>
57 <TR VALIGN=BASELINE>
58 <TH ALIGN="LEFT"><P>Cache Tcl Scripts:</TH><TD >
59 <SELECT NAME="Cache Tcl Scripts">
60 <OPTION SELECTED> On
61 <OPTION> Off
62 </SELECT>
63 </TD></TR>
64 <TR VALIGN=BASELINE>
65 <TH ALIGN="LEFT"><P>Tcl Trace Enabled:</TH><TD >
66 <SELECT NAME="Tcl Trace Enabled">
67 <OPTION SELECTED> Off
68 <OPTION> On
69 </SELECT>
70 </TD></TR>
```

Fig 6(a)

hamsetup.html

```
71 <TR VALIGN=BASELINE>
72 <TH ALIGN="LEFT"><P>Maximum Size of Synthesized Page:</TH><TD >
73 <INPUT NAME="Maximum Size of Synthesized Page"
74 TYPE=TEXT VALUE="8192">
75 </TD></TR>
76 <TR><TH ALIGN=LEFT><P><INPUT TYPE="reset" VALUE="RESET"></TH>
77 <TD ALIGN=LEFT><INPUT TYPE="submit" VALUE="APPLY"></TD></TR>
78 </TABLE>
79 </FORM>
80 <P>Go to <A HREF="http://mad.hprc.tandem.com:8080/ham/cget/TCP%3a168.87.28.7%2f806
   +2%3aHAM%2fHyperMedia%20Adapter%20NSK%2fConfiguration/"><STRONG>Browse-Only Mode</ST
   +RONG></A> of
81 <STRONG>Configuration</STRONG>
82 <P>
83 <P><CENTER><IMG SRC="/icons/pcbrblu.gif"></CENTER><P>
84 <BR><A HREF="http://mad.hprc.tandem.com:8080/ham/get/" TARGET="_top">
85 <IMG SRC="/icons/icback.gif"> HAM Home Page</A>
86 </BODY></HTML>
```

Fig 6(b)

Netscape - [Source of: http://mad.hpre.tandem.com:8080/hsm/get/TCP%3a168.87.28.7%3a8043%3aHAM]
1 <HTML>
2 <HEAD>
3 <TITLE>HyperMedia Adapter NSKHTCP:168.87.28.7/8043:HAM </TITLE>
4 <BASE HREF="http://mad.hpre.tandem.com:8080/hsm/get/TCP%3a168.87.28.7%3a8043%3aHAM
%3fHyperMedia%20Adapter%20NSK/">
5 </HEAD>
6
7 <BODY BACKGROUND="/icons/testpty.gif">
8 <TABLE WIDTH="100%" HEIGHT="100%" ALIGN="CENTER" |
BORDER="0">
10 <TR><TH><FONT SIZE=5>HyperMedia Adapter NSK</FONT></TH></TR>
11 </TABLE></BODY></HTML>
702

Fig 7(a)

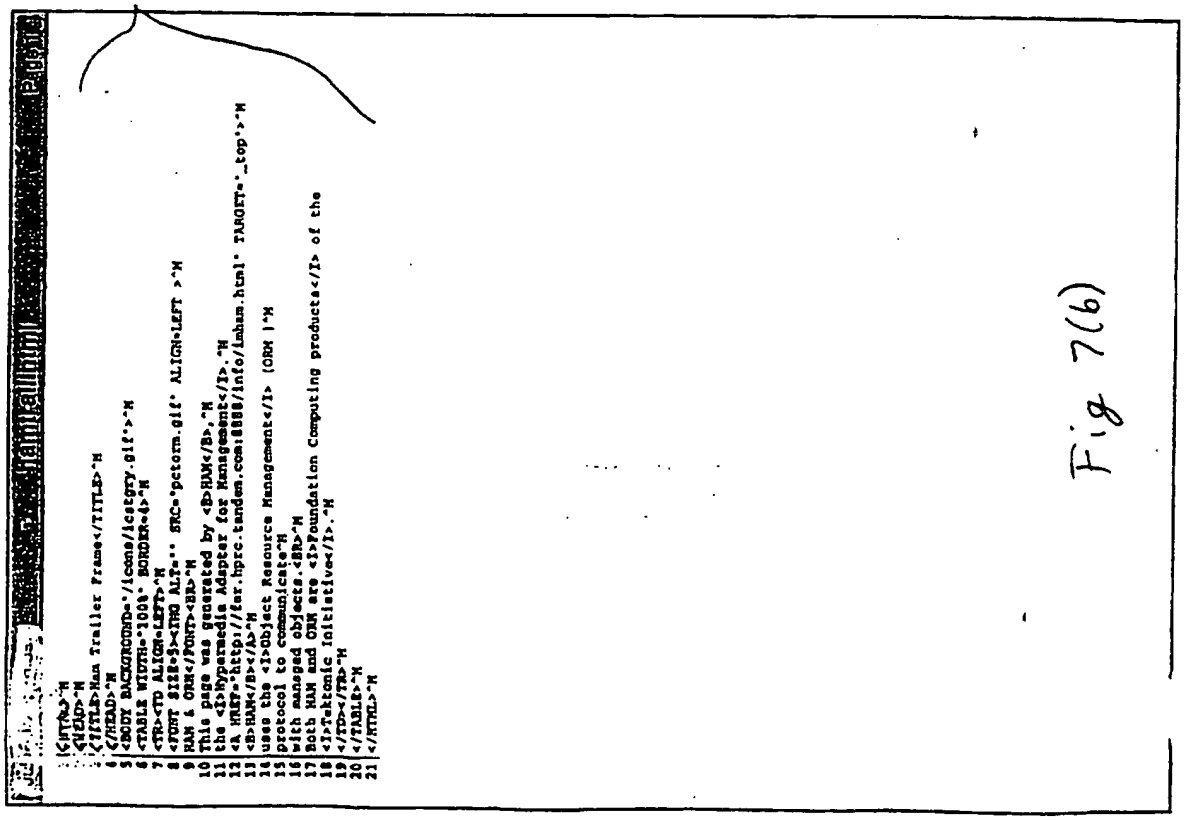

704
<HTML>^M
<HEAD>^M
<TITLE>Man Trailer Frame</TITLE>^M
</HEAD>^M
<BODY BACKGROUND='/icons/icatgry.gif'>^M
<TABLE WIDTH='100%' BORDER=4>^M
<TR><TD ALIGN=LEFT>^M
<FONT SIZE=5><IMG ALT='' SRC='pctcm.gif' ALIGN=LEFT >^M
MAN & ORK</FONT>
^M
This page was generated by <B>HAM</B>.^M
the <I>hypermedia Adapter for Management</I>.^M
<A HREF='http://far.hprc.tandem.com:8888/info/lnhmm.html' TARGET='_top'>^M
<B>HAM</B></A>^M
uses the <I>Object Resource Management</I> (ORM)^M
protocol to communicate^M
with managed objects.
^M
Both HAM and ORM are <I>Foundation Computing products</I> of the
<I>Tektronic Initiative</I>.^M
</TD></TR>^M
</TABLE>^M
</HTML>^M

Fig 7(b)

706
<HTML>
<HEAD>
<TITLE>Index into HyperMedia Adapter HSKHTCP:168.87.28.7/8062:HAM </TITLE>
<BASE HREF="http://mad.hprc.tandem.com:8080/ham/cgat/TCP%3a168.87.28.7%2f8062%3aHAM
%2fHyperMedia%20Adapter%20HSK/">
</HEAD>
<BODY BACKGROUND='/icons/icatfly.gif'>
<H3><A HREF="http://mad.hprc.tandem.com:8080/ham/cgat/TCP%3a168.87.28.7%2f8062%3aH
AM%2fHyperMedia%20Adapter%20HSK%2fProduct%20Information//" TARGET="hamcontent">
<IMG ALT="" SRC='/icons/icbox.gif'>
Product Information </A></H3>
<H3><A HREF="http://mad.hprc.tandem.com:8080/ham/cgat/TCP%3a168.87.28.7%2f8062%3aH
AM%2fHyperMedia%20Adapter%20HSK%2fInstance%20Information//" TARGET="hamcontent">
<IMG ALT="" SRC='/icons/iclnstc.gif'>
Instance Information </A></H3>
<H3><A HREF="http://mad.hprc.tandem.com:8080/ham/cgat/TCP%3a168.87.28.7%2f8062%3aH
AM%2fHyperMedia%20Adapter%20HSK%2fConfiguration//" TARGET="hamcontent">
<IMG ALT="" SRC='/icons/icconfig.gif'>
Configuration </A></H3>
<H3><A HREF="http://mad.hprc.tandem.com:8080/ham/cgat/TCP%3a168.87.28.7%2f8062%3aH
AM%2fHyperMedia%20Adapter%20HSK%2fStatistics//" TARGET="hamcontent">
<IMG ALT="" SRC='/icons/icstats.gif'>
Statistics </A></H3>
<P>
<P><IMG SRC='/icons/pcabble.gif'><P>
<P><A HREF="http://mad.hprc.tandem.com:8080/ham/gat/" TARGET="_top">
<IMG SRC='/icons/icback.gif'> HAM Home Page</A>
</BODY></HTML>

Fig 7(c)

ORM Protocol examples

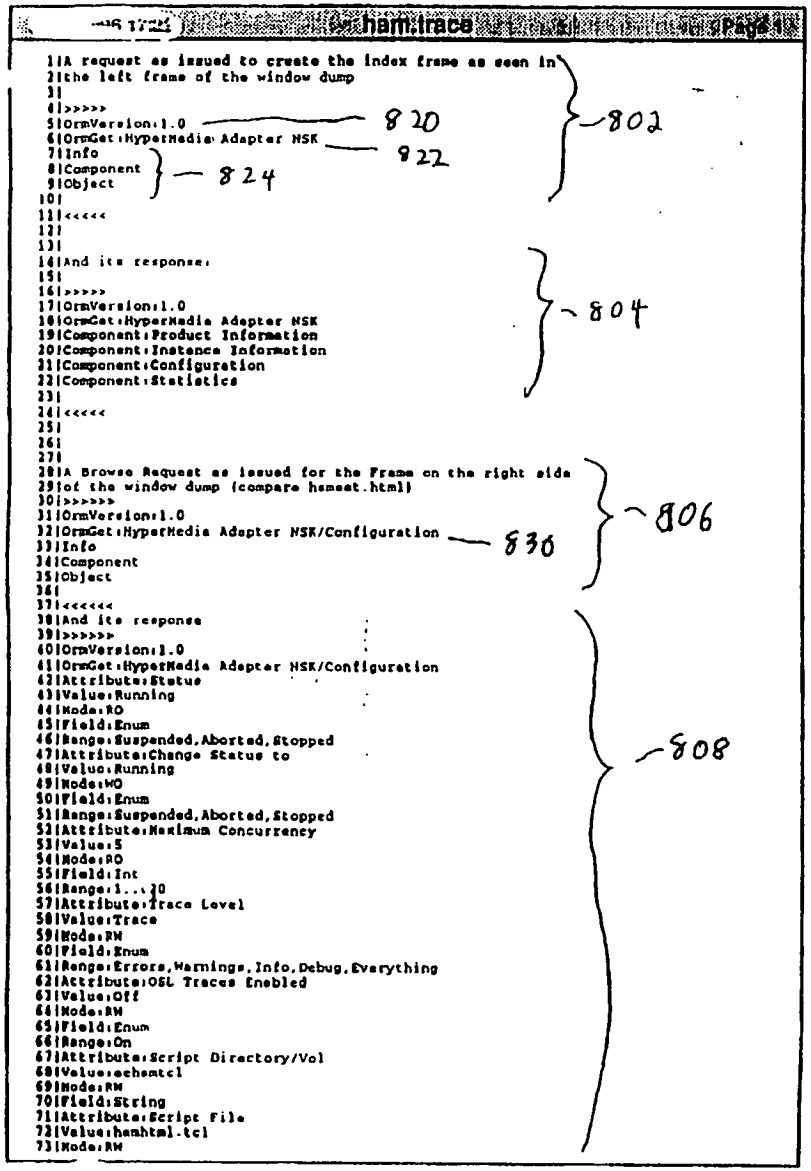


Fig. 8(a)

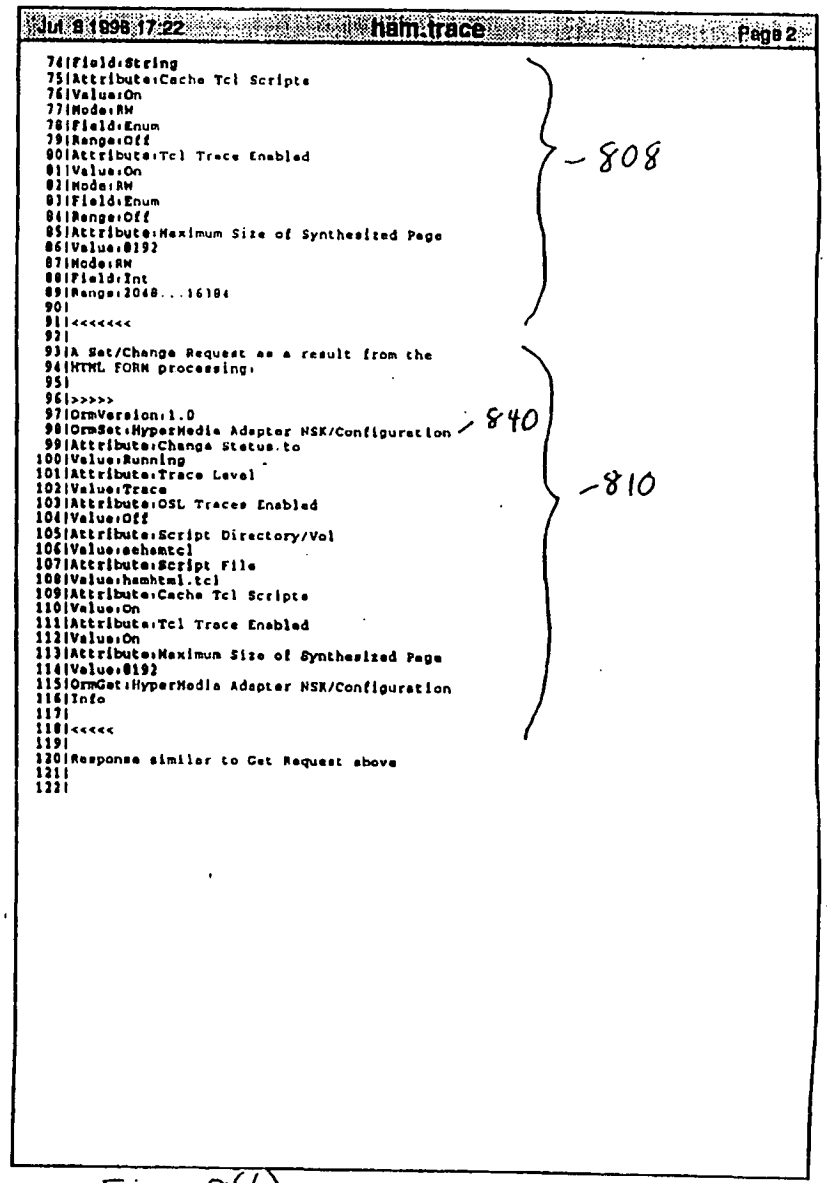


Fig. 8(b)

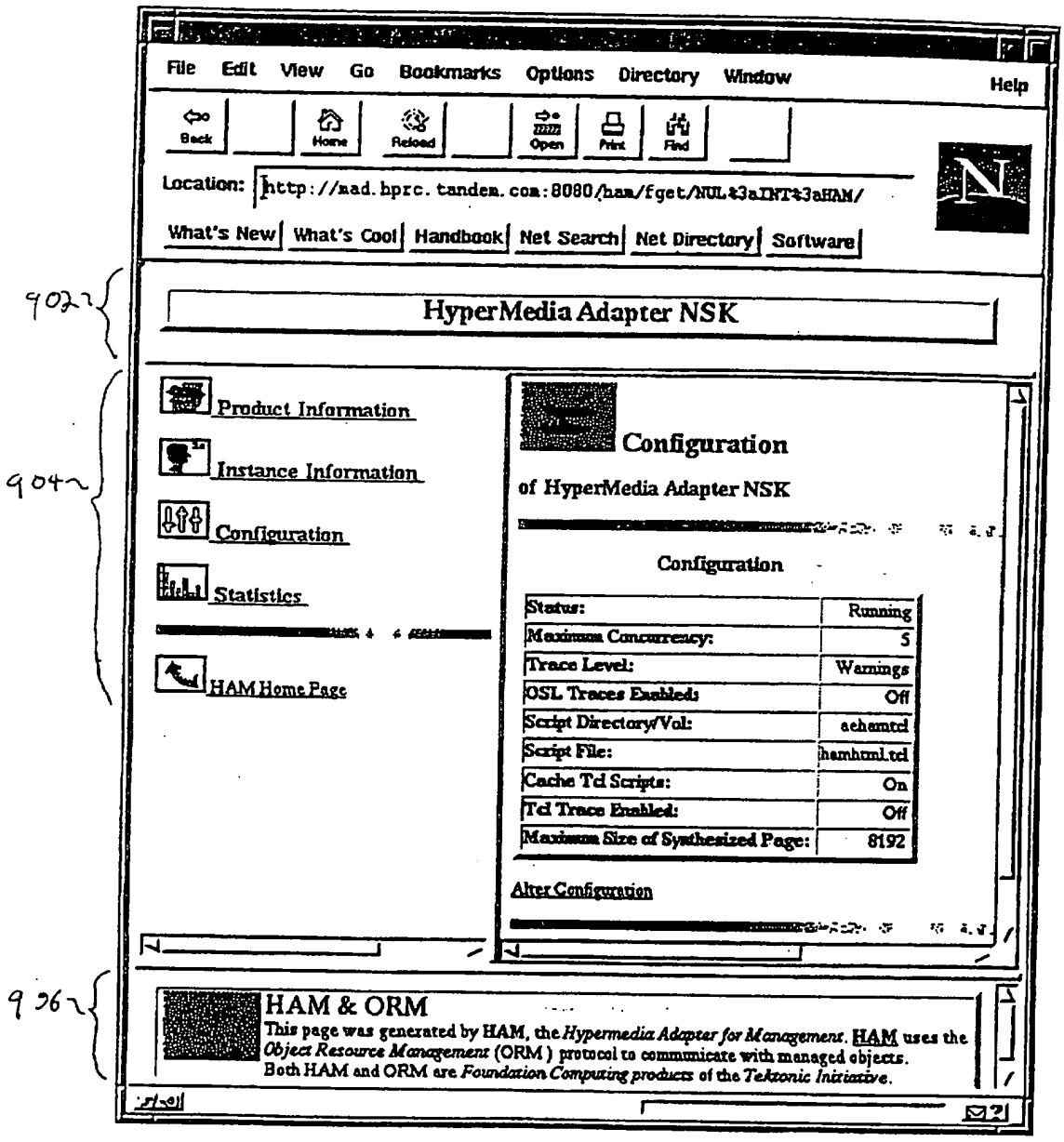
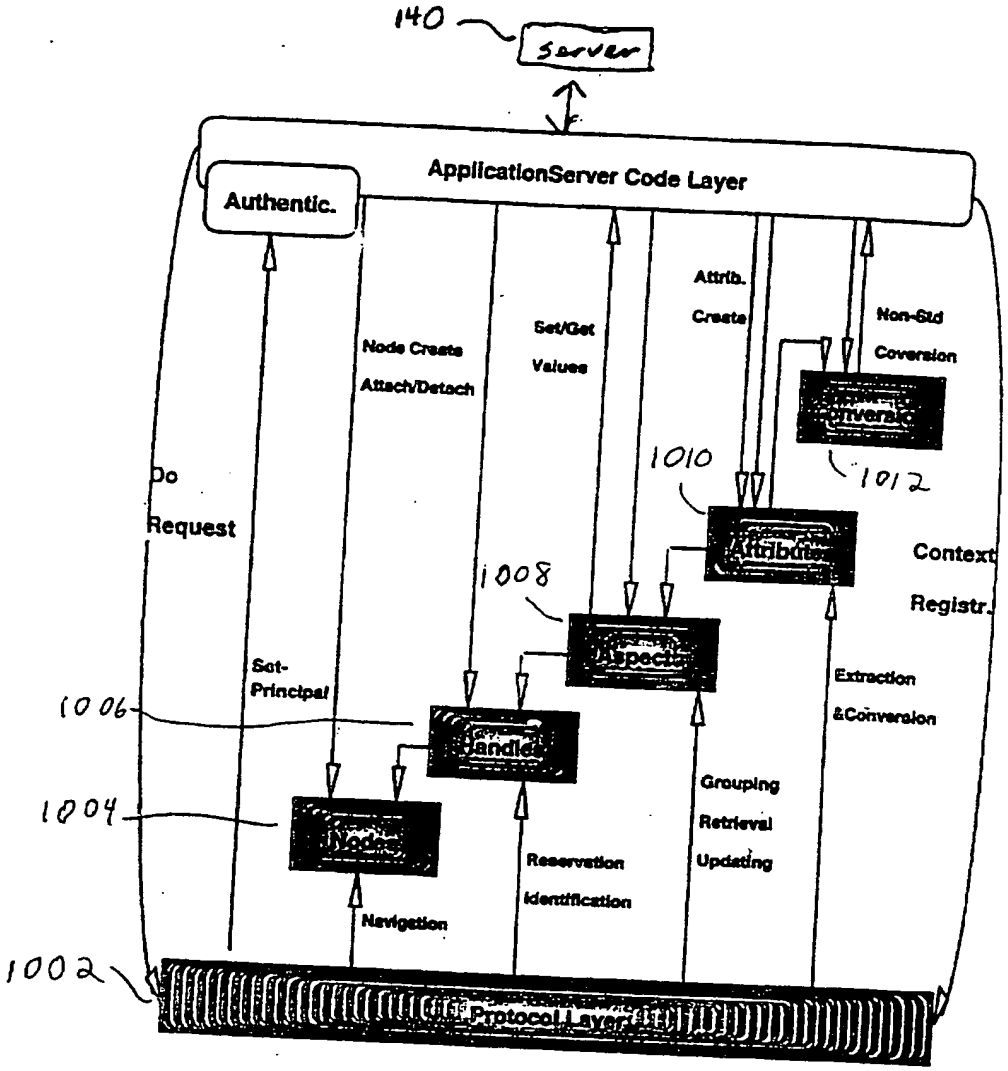

File Edit View Go Bookmarks Options Directory Window Help
Back Home Reload Open Print Find
Location: http://nad.hprc.tandem.com:8080/ham/fget/NUL%3aINT%3aHAM/
What's New What's Cool Handbook Net Search Net Directory Software
902
HyperMedia Adapter NSK
904
Product Information
Instance Information
Configuration
Statistics
HAM Home Page
Configuration
of HyperMedia Adapter NSK
Configuration
Status: Running
Maximum Concurrency: 5
Trace Level: Warnings
OSL Traces Enabled: Off
Script Directory/Vol: achamtcl
Script File: hamhtml.tcl
Cache Tcl Scripts: On
Tcl Trace Enabled: Off
Maximum Size of Synthesized Page: 8192
Alter Configuration
906
HAM & ORM
This page was generated by HAM, the Hypermedia Adapter for Management. HAM uses the Object Resource Management (ORM) protocol to communicate with managed objects.
Both HAM and ORM are Foundation Computing products of the Tektonic Initiative.

Fig. 9



The Layered Structure of the ORM Support Library

Fig. 10

INTERNATIONAL SEARCH REPORT

Inter ...nal Application No
PCT/US 97/11885

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F17/30

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	JAGANNATHAN V ET AL: "COLLABORATIVE INFRASTRUCTURES USING THE WWW AND CORBA-BASED ENVIRONMENTS" PROCEEDINGS - THE WORKSHOP ON ENABLING TECHNOLOGIES: INFRASTRUCTURE FOR COLLABORATIVE ENTERPRISES, 19 June 1996, pages 292-297, XP000645510 see page 293, column 1, line 39 - page 294, column 1, line 5 -----	1,14

☐ Further documents are listed in the continuation of box C.

☐ Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

1

Date of the actual completion of the international search	Date of mailing of the international search report
6 November 1997	12. 11. 97
Name and mailing address of the ISA European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Authorized officer Katerbau, R

Form PCT/ISA/210 (second sheet) (July 1992)